quanteda

February 2, 2015

Type Package

Title Quantitative Analysis of Textual Data

Version 0.7.0.000

Date 2015-01-29

Author Ken Benoit <kbenoit@lse.ac.uk> and Paul Nulty <p.nulty@lse.ac.uk>

Maintainer Ken Benoit <kbenoit@lse.ac.uk>

Description A package for the management and quantitative analysis of textual data with R. quanteda makes it easy to manage texts in the form of a corpus, defined as a collection of texts that includes document-level variables specific to each text, as well as meta-data for documents and for the collection as a whole. quanteda includes tools to make it easy and fast to manuipulate the texts the texts in a corpus, for instance by tokenizing them, with or without stopwords or stemming, or to segment them by sentence or paragraph units. quanteda implements bootstrapping methods for texts that makes it easy to resample texts from pre-defined units, to facilitate computation of confidence intervals on textual statistics using techniques of non-parametric bootstrapping, but applied to the original texts as data. quanteda includes a suite of sophisticated tools to extract features of the texts into a quantitative matrix, where these features can be defined according to a dictionary or thesaurus, including the declaration of collocations to be treated as single features. Once converted into a quantitative matrix (known as a ``dfm" for document-feature matrix), the textual features can be analyzed using quantitative methods for describing, comparing, or scaling texts, or topic modelling, or used to train machine learning methods for class prediction.

License GPL-3

Depends R (>= 3.0)

Imports methods,
Matrix (>= 1.1),
data.table (>= 1.9.3),
SnowballC,
wordcloud,
slam,
tm,
proxy,
ca,
stm,

topicmodels,
streamR,
jsonlite,
httr,
austin

Suggests quantedaData,
entropy,
openNLP,
RJSONIO,
RCurl,
twitteR,
XML,
lda,
tcltk2,
knitr,
rjags,
coda,
lattice,
xlsx

URL http://github.com/kbenoit/quanteda

BugReports https://github.com/kbenoit/quanteda/issues

LazyData TRUE

VignetteBuilder knitr

R topics documented:

quanteda-package 4
bigrams 4
changeunits 5
clean 6
collocations 7
colSums,dfmSparse-method 8
compoundWords 9
corpus 10
countSyllables 13
describeTexts 14
dfm 14
dfm-class 17
dfm2ldaformat 20
dfm2stmformat 21
dfm2tmformat 21
directory 22
docfreq 23
docnames 23
docvars 24
encoding 25
excel 26
features 26
flatten.dictionary 27
getFBpage 28

getRootFileNames . . . 29
getTextDir . . . 29
getTextDirGui . . . 30
getTextFiles . . . 30
getTimeline . . . 31
getTweets . . . 32
inaugCorpus . . . 33
json . . . 33
kwic . . . 34
language . . . 35
MCMCirtPoisson1d . . . 35
metacorpus . . . 37
metadoc . . . 38
ndoc . . . 39
ngrams . . . 39
nresample . . . 40
plot.dfm . . . 41
predict.textmodel . . . 42
print,dfmSparse-method . . . 43
print.textmodel . . . 44
readLIWCdict . . . 44
readWStatDict . . . 45
resample . . . 46
segmentSentence . . . 47
settings . . . 49
settingsInitialize . . . 50
similarity . . . 50
sort.dfm . . . 51
statLexdiv . . . 52
stopwords . . . 54
stopwordsGet . . . 55
stopwordsRemove . . . 55
subset.corpus . . . 56
summary.corpus . . . 57
summary.textmodel . . . 58
syllableCounts . . . 58
textmodel . . . 59
textmodel2 . . . 60
textmodel2_wordscores . . . 61
textmodel_ca . . . 63
textmodel_fitted-class . . . 64
textmodel_lda . . . 64
textmodel_NB . . . 65
textmodel_wordfish . . . 67
textmodel_wordscores . . . 68
texts . . . 69
tokenize . . . 69
topfeatures . . . 70
trim . . . 71
trimdfm . . . 72
uk2010immig . . . 73
weight . . . 73

wordfishMCMC 75
wordstem 77
zipfiles 78

Index **79**

`quanteda-package` *An R package for the quantitative analysis of textual data.*

Description

A set of functions for creating and managing text corpora, extracting features from text corpora, and analyzing those features using quantitative methods.

More detailed description, and some examples, to go here.

Author(s)

Ken Benoit and Paul Nulty

`bigrams` *Create bigrams*

Description

Create bigrams

Usage

```
bigrams(text, window = 1, concatenator = "_", include.unigrams = FALSE,
  ignoredFeatures = NULL, skipGrams = FALSE, ...)
```

Arguments

`text`	character vector containing the texts from which bigrams will be constructed
`window`	how many words to be counted for adjacency. Default is 1 for only immediately neighbouring words. This is only available for bigrams, not for ngrams.
`concatenator`	character for combining words, default is _ (underscore) character
`include.unigrams`	if `TRUE`, return unigrams as well
`ignoredFeatures`	a character vector of features to ignore
`skipGrams`	If `FALSE` (default), remove any bigram containing a feature listed in `ignoredFeatures`, otherwise, first remove the features in `ignoredFeatures`, and then create bigrams. This means that some "bigrams" will actually not occur as adjacent features in the original text. See examples.
`...`	provides additional arguments passed to tokenize

Value

a character vector of bigrams

Author(s)

Ken Benoit and Kohei Watanabe

Examples

```
bigrams("The quick brown fox jumped over the lazy dog.")
bigrams(c("The quick brown fox", "jumped over the lazy dog."))
bigrams(c("The quick brown fox", "jumped over the lazy dog."), window=2)
bigrams(c("I went to tea with her majesty Queen Victoria.", "Does tea have extra caffeine?"))
bigrams(c("I went to tea with her majesty Queen Victoria.", "Does tea have extra caffeine?"),
        ignoredFeatures=stopwords$english)
bigrams(c("I went to tea with her majesty Queen Victoria.", "Does tea have extra caffeine?"),
        ignoredFeatures=stopwords$english, skipGrams=TRUE)
```

---

changeunits *change the document units of a corpus*

Description

For a corpus, recast the documents down or up a level of aggregation. "Down" would mean going from documents to sentences, for instance. "Up" means from sentences back to documents. This makes it easy to reshape a corpus from a a collection of documents into a collection of sentences, for instance.

Usage

add a current units attribute

```
changeunits(corp, to = c("sentences", "paragraphs", "documents"), ...)
```

Arguments

corp	corpus whose document units will be reshaped
to	new documents units for the corpus to be recast in
...	passes additional arguments to segment

Examples

```
# simple example
mycorpus <- corpus(c(textone="This is a sentence.  Another sentence.  Yet another.",
                     textwo="Premiere phrase.  Deuxieme phrase."),
                   docvars=list(country=c("UK", "USA"), year=c(1990, 2000)),
                   notes="This is a simple example to show how changeunits() works.")
language(mycorpus) <- c("english", "french")
summary(mycorpus)
summary(changeunits(mycorpus, to="sentences"), showmeta=TRUE)

# example with inaugural corpus speeches
mycorpus2 <- subset(inaugCorpus, Year>2004)
mycorpus2
```

```
paragCorpus <- changeunits(mycorpus2, to="paragraphs")
paragCorpus
summary(paragCorpus, 100, showmeta=TRUE)
## Note that Bush 2005 is recorded as a single paragraph because that text used a single
## \n to mark the end of a paragraph.
```

---

clean *simple cleaning of text before processing*

Description

clean removes punctuation and digits from text, using the regex character classes for punctuation and digits. clean uses the standard R function tolower to convert the text to lower case. Each of these steps is optional, but switched on by default, so for example, to remove punctuation and convert to lower, but keep digits, the command would be: clean(mytexts, removeDigits=FALSE)

Usage

```
clean(x, ...)

## S3 method for class 'character'
clean(x, removeDigits = TRUE, removePunct = TRUE,
  lower = TRUE, additional = NULL, twitter = TRUE, removeURL = TRUE,
  ...)

## S3 method for class 'corpus'
clean(x, removeDigits = TRUE, removePunct = TRUE,
  lower = TRUE, additional = NULL, twitter = TRUE, ...)
```

Arguments

x	The object to be cleaned. Can be either a character vector or a corpus object. If x is a corpus, clean returns a copy of the x with the texts cleaned.
...	additional parameters
removeDigits	remove numbers if TRUE
removePunct	remove punctuation if TRUE
lower	convert text to lower case TRUE
additional	additional characters to remove (regular expression)
twitter	if TRUE, do not remove @ or #
removeURL	removes URLs (web addresses starting with http: or https:), based on a regular expression from http://daringfireball.net/2010/07/improved_regex_for_matching_urls

Value

A character vector equal in length to the original texts, after cleaning.

Examples

```
clean("This is 1 sentence with 2.0 numbers in it, and one comma.", removeDigits=FALSE)
clean("This is 1 sentence with 2.0 numbers in it, and one comma.", lower=FALSE)
clean("We are his Beliebers, and him is #ourjustin @justinbieber we love u", twitter=TRUE)
clean("Collocations can be represented as inheritance_tax using the _ character.")
clean("But under_scores can be removed using the additional argument.", additional="[_]")
clean("This is a $1,500,000 budget and $20bn cash and a $5 cigar.")
clean("This is a $1,500,000 budget and $20bn cash and a $5 cigar.", removeDigits=FALSE)
clean("URL regex from http://daringfireball.net/2010/07/improved_regex_for_matching_urls.")

# for a vector of texts
clean(c("This is 1 sentence with 2.0 numbers in it, and one comma.",
        "$1.2 billion was spent on text analysis in 2014."))
```

`collocations` *Detect collocations from text*

Description

Detects collocations (currently, bigrams and trigrams) from texts or a corpus, returning a data.frame of collocations and their scores, sorted in descending order of the association measure.

Usage

```
collocations(x, ...)

## S3 method for class 'character'
collocations(x, method = c("lr", "chi2", "pmi", "dice",
  "all"), n = 2, top = NULL, ...)

## S3 method for class 'corpus'
collocations(x, method = c("lr", "chi2", "pmi", "dice",
  "all"), n = 2, top = NULL, ...)
```

Arguments

`x`	a text, a character vector of texts, or a corpus
`...`	additional parameters passed to clean
`method`	association measure for detecting collocations. Available measures are:

`"lr"` The likelihood ratio statistic G^2, computed as:

$$2 * \sum_i \sum_j (n_{ij} * log \frac{n_{ij}}{m_{ij}}$$

`"chi2"` Pearson's χ^2 statistic, computed as:

$$\sum_i \sum_j \frac{(n_{ij} - m_{ij})^2}{m_{ij}}$$

`"pmi"` point-wise mutual information score, computed as log $n_{11}/m11$

`"dice"` the Dice coefficient, computed as $n_{11}/n_{1.} + n_{.1}$

	`"all"` returns all of the above
`n`	length of the collocation. Only bigram (`n=2`) and trigram (`n=3`) collocations are implemented so far.
`top`	the number of collocations to return, sorted in descending order of the requested statistic, or G^2 if none is specified.

Value

A data.table of collocations, their frequencies, and the computed association measure(s).

Author(s)

Kenneth Benoit

References

McInnes, B T. 2004. "Extending the Log Likelihood Measure to Improve Collocation Identification." M.Sc. Thesis, University of Minnesota.

See Also

bigrams, ngrams

Examples

```
collocations(inaugTexts, top=10)
collocations(inaugCorpus, top=10, method="all")
collocations(inaugTexts, top=10, n=3)
collocations(inaugCorpus, top=10, method="all", n=3)
```

---

colSums,dfmSparse-method

the calculations myDfm <- dfm(inaugTexts, verbose=FALSE)

---

put with rowSums

Description

the calculations myDfm <- dfm(inaugTexts, verbose=FALSE)

Usage

```
## S4 method for signature 'dfmSparse'
colSums(x, na.rm = FALSE, dims = 1L, ...)
```

Arguments

`x`	a dfm, inheriting from Matrix
`na.rm`	if `TRUE`, omit missing values (including `NaN`) from the calculations
`dims`	ignored
`...`	additional arguments, for methods/generic compatibility

compoundWords *convert phrases into single tokens*

Description

Replace multi-word phrases in text(s) with a compound version of the phrases concatenated with connector (by default, the "_" character) to form a single token. This prevents tokenization of the phrases during subsequent processing by eliminating the whitespace delimiter.

Usage

```
compoundWords(txts, dictionary, connector = "_")
```

Arguments

txts	character or character vector of texts
dictionary	a list or named list (such as a quanteda dictionary) that contains some phrases, defined as multiple words delimited by whitespace. These can be up to 9 words long.
connector	the concatenation character that will connect the words making up the multi-word phrases. The default _ is highly recommended since it will not be removed during normal cleaning and tokenization (while nearly all other punctuation characters, at least those in the POSIX class [[:punct:]]) will be removed.

Value

character or character vector of texts with phrases replaced by compound "words" joined by the connector

Examples

```
mytexts <- c("The new law included a capital gains tax, and an inheritance tax.",
             "New York City has raised a taxes: an income tax and a sales tax.")
mydict <- list(tax=c("tax", "income tax", "capital gains tax", "inheritance tax"))
(cw <- compoundWords(mytexts, mydict))
print(dfm(cw), show.values=TRUE)

# when used as a dictionary for dfm creation
mydfm2 <- dfm(cw, dictionary=lapply(mydict, function(x) gsub(" ", "_", x)))
print(mydfm2, show.values=TRUE)
# to pick up "taxes" in the second text, set regular_expression=TRUE
mydfm3 <- dfm(cw, dictionary=lapply(mydict, function(x) gsub(" ", "_", x)),
              dictionary_regex=TRUE)
print(mydfm3, show.values=TRUE)
```

corpus *Constructor for corpus objects*

Description

Creates a corpus from a document source. The current available document sources are:

- a character vector (as in R class `char`) of texts;
- a directory of text files, using directory;
- a directory constructed from a zip file consisting of text files, using zipfiles; and
- a **tm** VCorpus class corpus object, meaning that anything you can use to create a **tm** corpus, including all of the tm plugins plus the built-in functions of tm for importing pdf, Word, and XML documents, can be used to create a quanteda corpus.

Corpus-level meta-data can be specified at creation, containing (for example) citation information and notes, as can document-level variables and document-level meta-data.

Usage

```
corpus(x, ...)

## S3 method for class 'directory'
corpus(x, enc = NULL, docnames = NULL,
  docvarsfrom = c("none", "filenames", "headers"), docvarnames = NULL,
  sep = "_", pattern = "\\.txt$", source = NULL, notes = NULL,
  citation = NULL, ...)

## S3 method for class 'excel'
corpus(x, docnames = row.names(x), textCol = 1,
  docvarsfrom = NULL, source = NULL, notes = NULL, citation = NULL, ...)

## S3 method for class 'twitter'
corpus(x, enc = NULL, notes = NULL, citation = NULL,
  ...)

## S3 method for class 'facebook'
corpus(x, enc = NULL, notes = NULL, citation = NULL,
  ...)

## S3 method for class 'VCorpus'
corpus(x, enc = NULL, notes = NULL, citation = NULL,
  ...)

## S3 method for class 'character'
corpus(x, enc = NULL, docnames = NULL, docvars = NULL,
  source = NULL, notes = NULL, citation = NULL, ...)

is.corpus(x)

## S3 method for class 'corpus'
c1 + c2
```

Arguments

x	A source of texts to form the documents in the corpus. This can be a filepath to a directory containing text documents (see directory), or a character vector of texts.
...	additional arguments
enc	A string specifying the input encoding for texts in the corpus. Must be a valid entry in iconvlist(), since the code in corpus.character will convert this to UTF-8 using iconv. Currently only one input encoding can be specified for a collection of input texts, meaning that you should not mix input text encoding types in a single corpus call.
docnames	Names to be assigned to the texts, defaults to the names of the character vector (if any), otherwise assigns "text1", "text2", etc.
docvarsfrom	Argument to specify where docvars are to be taken, from parsing the filenames separated by sep or from meta-data embedded in the text file header (headers).
docvarnames	Character vector of variable names for docvars
sep	Separator if docvars names are taken from the filenames.
pattern	filename extension - set to "*" if all files are desired. This is a regular expression. ? or glob?
source	A string specifying the source of the texts, used for referencing.
notes	A string containing notes about who created the text, warnings, To Dos, etc.
citation	Information on how to cite the corpus.
textCol	The column of the sheet that contains the texts the docvars from. By defauls, takes everything except the textCol by sep or from meta-data embedded in the text file header (headers).
docvars	A data frame of attributes that is associated with each text.
c1	corpus one to be added
c2	corpus two to be added

maybe a list for metadata?

Details

The + operator for a corpus object will combine two corpus objects, resolving any non-matching docvars or metadoc fields by making them into NA values for the corpus lacking that field. Corpus-level meta data is concatenated, except for source and notes, which are stamped with information pertaining to the creation of the new joined corpus.

There are some issues that need to be addressed in future revisions of quanteda concerning the use of factors to store document variables and meta-data. Currently most or all of these are not recorded as factors, because we use stringsAsFactors=FALSE in the data.frame calls that are used to create and store the document-level information, because the texts should always be stored as character vectors and never as factors.

Value

A corpus class object containing the original texts, document-level variables, document-level meta-data, corpus-level metadata, and default settings for subsequent processing of the corpus. A corpus consists of a list of elements described below, although these should only be accessed through accessor and replacement functions, not directly (since the internals may be subject to change). The structure of a corpus classed list object is:

`$documents`	A data frame containing the document level information, consisting of `texts`, user-named `docvars` variables describing attributes of the documents, and `metadoc` document-level metadata whose names begin with an underscore character, such as `_language`.
`$metadata`	A named list set of corpus-level meta-data, including `source` and `created` (both generated automatically unless assigned), `notes`, and `citation`.
`$settings`	Settings for the corpus which record options that govern the subsequent processing of the corpus when it is converted into a document-feature matrix (dfm). See settings.
`$tokens`	An indexed list of tokens and types tabulated by document, including information on positions. Not yet fully implemented.

make this flexible

`is.corpus` returns `TRUE` if the object is a corpus

Note

When `x` is a VCorpus object, the fixed metadata fields from that object are imported as document-level metadata. Currently no corpus-level metadata is imported, but we will add that soon.

See Also

docvars, metadoc, metacorpus, language, encoding, settings, texts

Examples

```
## Not run:
# import texts from a directory of files
summary(corpus(directory("~/Dropbox/QUANTESS/corpora/ukManRenamed"),
               enc="UTF-8", docvarsfrom="filenames",
               source="Ken's UK manifesto archive",
               docvarnames=c("Country", "Level", "Year", "language")), 5))
summary(corpus(directory("~/Dropbox/QUANTESS/corpora/ukManRenamed"),
               enc="UTF-8", docvarsfrom="filenames",
               source="Ken's UK manifesto archive",
               docvarnames=c("Country", "Level", "Year", "language", "Party")), 5))

# choose a directory using a GUI
corpus(directory())

# from a zip file on the web
myzipcorp <- corpus(zipfiles("http://kenbenoit.net/files/EUcoalsubsidies.zip"),
                    notes="From some EP debate about coal mine subsidies")
docvars(myzipcorp, speakername=docnames(myzipcorp))
summary(myzipcorp)

## End(Not run)
## Not run:

#
## import a tm VCorpus
if (require(tm)) {
    data(crude)    # load in a tm example VCorpus
    mytmCorpus <- corpus(crude)
    summary(mytmCorpus, showmeta=TRUE)
}
```

```
#
# create a corpus from texts
corpus(inaugTexts)

# create a corpus from texts and assign meta-data and document variables
uk2010immigCorpus <- corpus(uk2010immig,
                            docvars=data.frame(party=names(uk2010immig)),
                            enc="UTF-8")
```

`countSyllables` *Returns a count of the number of syllables in the input*

---

how about just syllables(x)

Description

This function takes a text and returns a count of the number of syllables it contains. For British English words, the syllable count is exact and looked up from the CMU pronunciation dictionary. For any word not in the dictionary the syllable count is estimated by counting vowel clusters.

Usage

```
countSyllables(sourceText)
```

Arguments

`sourceText`	Character vector of texts whose syllables will be counted

Details

This only works for English.

Value

numeric Named vector of counts of the number of syllables for each element of sourceText. When a word is not available in the lookup table, its syllables are estimated by counting the number of (English) vowels in the word.

Examples

```
countSyllables("This is an example sentence.")
myTexts <- c("Text one.", "Superduper text number two.", "One more for the road.")
names(myTexts) <- paste("myText", 1:3, sep="")
countSyllables(myTexts)
```

replace with a summary.character() method

describeTexts *print a summary of texts*

Description

Prints to the console a desription of the texts, including number of types, tokens, and sentences

Usage

```
describeTexts(txts, verbose = TRUE)
```

Arguments

txts	The texts to be described
verbose	Default is TRUE. Set to false to suppress output messages

Examples

```
describeTexts(c("testing this text", "and this one"))
describeTexts(uk2010immig)
```

dfm *create a document-feature matrix*

Description

Create a sparse matrix document-feature matrix from a corpus or a vector of texts. The sparse matrix construction uses the **Matrix** package, and is both much faster and much more memory efficient than the corresponding dense (regular `matrix`) representation. For details on the structure of the dfm class, see dfm-class.

Usage

```
dfm(x, ...)

## S3 method for class 'character'
dfm(x, verbose = TRUE, clean = TRUE, stem = FALSE,
  ignoredFeatures = NULL, keptFeatures = NULL, matrixType = c("sparse",
  "dense"), language = "english", fromCorpus = FALSE, bigrams = FALSE,
  thesaurus = NULL, dictionary = NULL, dictionary_regex = FALSE,
  addto = NULL, ...)

## S3 method for class 'corpus'
dfm(x, verbose = TRUE, clean = TRUE, stem = FALSE,
  ignoredFeatures = NULL, keptFeatures = NULL, matrixType = c("sparse",
  "dense"), language = "english", groups = NULL, bigrams = FALSE,
  thesaurus = NULL, dictionary = NULL, dictionary_regex = FALSE,
  addto = NULL, ...)
```

```
is.dfm(x)

as.dfm(x)

## S4 method for signature 'dfm'
as.matrix(x)

## S4 method for signature 'dfm'
as.data.frame(x)
```

Arguments

x	corpus or character vector from which to generate the document-feature matrix
...	additional arguments passed to clean
verbose	display messages if TRUE
clean	if FALSE, do no cleaning of the text
stem	if TRUE, stem words
ignoredFeatures	a character vector of user-supplied features to ignore, such as "stop words". Formerly, this was a Boolean option for stopwords = TRUE, but requiring the user to supply the list highlights the choice involved in using any stopword list. To access one possible list (from any list you wish), use the stopwordsGet() function or just (e.g.) stopwords$english.
keptFeatures	a use supplied regular expression defining which features to keep, while excluding all others. This can be used in lieu of a dictionary if there are only specific features that a user wishes to keep. To extract only Twitter user names hashtags, set keep = "@\w+\b" and make sure that twitter = TRUE as an additional argument passed to clean.
matrixType	if dense, produce a dense matrix; or it sparse produce a sparse matrix of class dgCMatrix from the **Matrix** package.
language	Language for stemming and stopwords. Choices are danish, dutch, english, finnish, french, for stemming, and SMART, danish, english, french, hungarian,norwegian, russian, swedi for stopwords.
fromCorpus	a system flag used internally, soon to be phased out.
bigrams	include bigrams as well as unigram features, if TRUE
thesaurus	A list of character vector "thesaurus" entries, in a dictionary list format, which can also include regular expressions if dictionary_regex is TRUE (see examples). Note that unlike dictionaries, each entry in a thesaurus key must be unique, otherwise only the first match in the list will be used. Thesaurus keys are converted to upper case to create a feature label in the dfm, as a reminder that this was not a type found in the text, but rather the label of a thesaurus key.
dictionary	A list of character vector dictionary entries, including regular expressions (see examples)
dictionary_regex	TRUE means the dictionary is already in regular expression format, otherwise it will be converted from "wildcard" format
addto	NULL by default, but if an existing dfm object is specified, then the new dfm will be added to the one named. If both dfm's are built from dictionaries, the combined dfm will have its Non_Dictionary total adjusted.
groups	Grouping variable for aggregating documents

Details

New as of v0.7: All dfms are by default sparse, a change from the previous behaviour. You can still create the older (S3) dense matrix type dfm object, but you will receive a disapproving warning message while doing so, suggesting you make the switch.

`is.dfm` returns `TRUE` if and only if its argument is a dfm.

`as.dfm` coerces a matrix or data.frame to a dfm

Value

A dfm-class object containing a sparse matrix representation of the counts of features by document, along with associated settings and metadata.

If you used `matrixType = "dense"` then the return is an old-style S3 matrix class object with additional attributes representing meta-data.

Author(s)

Kenneth Benoit

Examples

```
# with inaugural texts
(size1 <- object.size(dfm(inaugTexts, matrixType="sparse")))
(size2 <- object.size(dfm(inaugTexts, matrixType="dense")))
cat("Compacted by ", round(as.numeric((1-size1/size2)*100), 1), "%.\n", sep="")

# with stopwords English, stemming, and dense matrix
dfmsInaug2 <- dfm(inaugCorpus, ignoredFeatures = stopwordsGet(), stem=TRUE, matrixType="dense")

## with dictionaries
mycorpus <- subset(inaugCorpus, Year>1900)
mydict <- list(christmas=c("Christmas", "Santa", "holiday"),
               opposition=c("Opposition", "reject", "notincorpus"),
               taxing="taxing",
               taxation="taxation",
               taxregex="tax*",
               country="united states")
dictDfm <- dfm(mycorpus, dictionary=mydict)
dictDfm

## with the thesaurus feature
mytexts <- c("The new law included a capital gains tax, and an inheritance tax.",
             "New York City has raised a taxes: an income tax and a sales tax.")
mydict <- list(tax=c("tax", "income tax", "capital gains tax", "inheritance tax"))
dfm(compoundWords(mytexts, mydict), thesaurus=lapply(mydict, function(x) gsub("\\s", "_", x)))
# pick up "taxes" with "tax" as a regex
dfm(compoundWords(mytexts, mydict), thesaurus=list(anytax="tax"), dictionary_regex=TRUE)

## removing stopwords
testText <- "The quick brown fox named Seamus jumps over the lazy dog also named Seamus, with
             the newspaper from a a boy named Seamus, in his mouth."
testCorpus <- corpus(testText)
settings(testCorpus, "stopwords")
dfm(testCorpus, stopwords=TRUE)

## keep only certain words
```

```
dfm(testCorpus, keep="s$")  # keep only words ending in "s"
testTweets <- c("My homie @justinbieber #justinbieber shopping in #LA yesterday #beliebers",
             "2all the ha8ers including my bro #justinbieber #emabiggestfansjustinbieber",
             "Justin Bieber #justinbieber #belieber#fetusjustin #EMABiggestFansJustinBieber")
dfm(testTweets, keep="^#")  # keep only hashtags


## Not run:
# try it with approx 35,000 court documents from Lauderdale and Clark (200?)
load('~/Dropbox/QUANTESS/Manuscripts/Collocations/Corpora/lauderdaleClark/Opinion_files.RData')
txts <- unlist(Opinion_files[1])
names(txts) <- NULL

# dfms without cleaning
require(Matrix)
system.time(dfmsBig <- dfm(txts, clean=FALSE, verbose=FALSE))
object.size(dfmsBig)
dim(dfmsBig)
# compare with tm
require(tm)
tmcorp <- VCorpus(VectorSource(txts))
system.time(tmDTM <- DocumentTermMatrix(tmcorp))
object.size(tmDTM)
dim(tmDTM)

# with cleaning - the gsub() calls in clean() take a long time
system.time(dfmsBig <- dfm(txts, clean=TRUE, additional="[-_\\x{h2014}]"))
object.size(dfmsBig)
dim(dfmsBig)
# 100 top features
topf <- colSums(dfmsBig)
names(topf) <- colnames(dfmsBig)
head(sort(topf, decreasing=TRUE), 100)

## End(Not run)
# sparse matrix from a corpus
mydfms <- dfm(inaugCorpus, matrixType="sparse")
data(ie2010Corpus, package="quantedaData")
mydfms2 <- dfm(ie2010Corpus, groups = "party", matrixType="sparse")
```

`dfm-class` *Virtual class "dfm" for a document-feature matrix*

Description

The dfm class of object is a type of Matrix-class object with additional slots, described below. **quanteda** uses two subclasses of the `dfm` class, depending on whether the object can be represented by a sparse matrix, in which case it is a `dfmSparse` class object, or if dense, then a `dfmDense` object. See Details.

Usage

```
## S4 method for signature 'dfmSparse'
t(x)

## S4 method for signature 'dfmDense'
t(x)

## S4 method for signature 'dfm'
t(x)

## S3 method for class 'dfm'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'dfmDense,index,index,missing'
x[i = NULL, j = NULL, ...,
  drop = FALSE]

## S4 method for signature 'dfmDense,index,index,logical'
x[i = NULL, j = NULL, ...,
  drop = FALSE]

## S4 method for signature 'dfmDense,index,missing,missing'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'dfmDense,index,missing,logical'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'dfmDense,missing,index,missing'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'dfmDense,missing,index,logical'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'dfmDense,missing,missing,missing'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'dfmDense,missing,missing,logical'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'dfmSparse,index,index,missing'
x[i = NULL, j = NULL, ...,
  drop = FALSE]

## S4 method for signature 'dfmSparse,index,index,logical'
x[i = NULL, j = NULL, ...,
  drop = FALSE]

## S4 method for signature 'dfmSparse,index,missing,missing'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'dfmSparse,index,missing,logical'
x[i, j, ..., drop = FALSE]
```

```
## S4 method for signature 'dfmSparse,missing,index,missing'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'dfmSparse,missing,index,logical'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'dfmSparse,missing,missing,missing'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'dfmSparse,missing,missing,logical'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'dfmSparse,numeric'
e1 + e2

## S4 method for signature 'numeric,dfmSparse'
e1 + e2

## S4 method for signature 'dfmDense,numeric'
e1 + e2

## S4 method for signature 'numeric,dfmDense'
e1 + e2
```

Arguments

x	the sparse dfm
i	index for documents
j	index for features
...	additional arguments not used here
drop	always set to FALSE
e1	first quantity in "+" operation for dfm
e2	second quantity in "+" operation for dfm

Details

The dfm class is a virtual class that will contain one of two subclasses for containing the cell counts of document-feature matrixes: dfmSparse or dfmDense.

The dfmSparse class is a sparse matrix version of dfm-class, inheriting dgCMatrix-class from the **Matrix** package. It is the default object type created when feature counts are the object of interest, as typical text-based feature counts tend contain many zeroes. As long as subsequent transformations of the dfm preserve cells with zero counts, the dfm should remain sparse.

When the **Matrix** package implements sparse integer matrixes, we will switch the default object class to this object type, as integers are 4 bytes each (compared to the current numeric double type requiring 8 bytes per cell.)

The dfmDense class is a sparse matrix version of dfm-class, inheriting dgeMatrix-class from the **Matrix** package. dfm objects that are converted through weighting or other transformations into cells without zeroes will be automatically converted to the dfmDense class. This will necessarily be a much larger sized object than one of dfmSparse class, because each cell is recorded as a numeric (double) type requiring 8 bytes of storage.

Slots

`settings` settings that govern corpus handling and subsequent downstream operations, including the settings used to clean and tokenize the texts, and to create the dfm. See `settings`.

`weighting` the feature weighting applied to the dfm. Default is `"frequency"`, indicating that the values in the cells of the dfm are simple feature counts. To change this, use the `weight` method.

`smooth` a smoothing parameter, defaults to zero. Can be changed using either the `smooth` or the `weight` methods.

`Dimnames` These are inherited from Matrix-class but are named `docs` and `features` respectively.

Add reference to corpus object - so that if in memory, can reference it

---

`dfm2ldaformat` *Convert a dfm into the format needed by* **lda**

Change to convert.dfm(x, from="", to="")

Description

Convert a quanteda dfm object into the indexed format required by the topic modelling package **lda**.

Usage

```
dfm2ldaformat(d)
```

Arguments

`d`	A `dfm` object

Value

A list with components "documents" and "vocab" as needed by lda.collapsed.gibbs.sampler

Examples

```
mycorpus <- subset(inaugCorpus, Year>1970)
d <- dfm(mycorpus, stopwords=TRUE)
d <- trimdfm(d, minCount=5, minDoc=3)
td <- dfm2ldaformat(d)
if (require(lda)) {
    tmodel.lda <- lda.collapsed.gibbs.sampler(documents=td$documents,
                                              K=10,
                                              vocab=td$vocab,
                                              num.iterations=50, alpha=0.1, eta=0.1)
    top.topic.words(tmodel.lda$topics, 10, by.score=TRUE) # top five words in each topic
}
```

`dfm2stmformat` *convert a dfm to stm's input document format*

Description

Convert a quanteda dfm object into the indexed format needed for estimating a structural topic model from the **stm** package using stm.

Usage

```
dfm2stmformat(data)
```

Arguments

`data`	dfm object to be converted

Value

A list containing the following elements:

`documents`	A list containing the documents in the stm format.
`vocab`	Character vector of vocabulary.
`meta`	NULL

Note

Meta-data will need to be passed separately to stm as this information is not included in a dfm object.

Examples

```
mydfm <- dfm(inaugTexts)
mydfmStm <- dfm2stmformat(mydfm)
str(mydfmStm)
```

`dfm2tmformat` *Convert a dfm into a* **tm** *DocumentTermMatrix*

Description

tm represents sparse document-feature matrixes in the simple triplet matrix format of the package **slam**. This function converts a `dfm` into a `DocumentTermMatrix`, enabling a dfm to be used with other packages that expect this format, such as **topicmodels**.

Usage

```
dfm2tmformat(d, weighting = weightTf)
```

Arguments

d	A dfm object
weighting	weight function arguments passed to as.TermDocumentMatrix, defaults to term frequency (see as.DocumentTermMatrix for a list of options, such as tf-idf).

Value

A simple triplet matrix of class as.DocumentTermMatrix

Examples

```
mycorpus <- subset(inaugCorpus, Year>1970)
d <- trimdfm(dfm(mycorpus), minCount=5, minDoc=3)
dim(d)
td <- dfm2tmformat(d)
length(td$v)
if (require(topicmodels)) (tmodel.lda <- LDA(td, control = list(alpha = 0.1), k = 5))
```

---

directory *Function to declare a connection to a directory (containing files)*

Description

Function to declare a connection to a directory, although unlike file it does not require closing. If the directory does not exist, the function will return an error.

Usage

```
directory(path = NULL)
```

Arguments

path	String describing the full path of the directory or NULL to use a GUI to choose a directory from disk

Examples

```
## Not run:
# name a directory of files
mydir <- directory("~/Dropbox/QUANTESS/corpora/ukManRenamed")
corpus(mydir)

# choose a directory using a GUI
corpus(directory())
## End(Not run)
```

`docfreq` *get the document frequency of a feature*

Description

Returns the document frequency of a feature in a dfm-class object, which occurs greater than a threshold.

Usage

```
docfreq(object, threshold = 0)

## S4 method for signature 'dfmDense,numeric'
docfreq(object, threshold = 0)

## S4 method for signature 'dfmDense,missing'
docfreq(object, threshold = 0)

## S4 method for signature 'dfmSparse,numeric'
docfreq(object, threshold = 0)

## S4 method for signature 'dfmSparse,missing'
docfreq(object, threshold = 0)

## S4 method for signature 'dfm,numeric'
docfreq(object, threshold = 0)

## S4 method for signature 'dfm,missing'
docfreq(object, threshold = 0)
```

Is there any way to prevent all of these methods from being listed?

Arguments

`object`	a dfm-class document-feature matrix
`threshold`	numeric value of the threshold for counting a feature as existing in the document, default is 0

`docnames` *get or set document names*

Description

Extract the document names from a corpus or a document-feature matrix. Document names are the rownames of the documents data.frame in a corpus, or the rownames of the dfm object for a dfm. of the dfm object.

`docnames` queries the document names of a corpus or a dfm

`docnames <-` assigns new values to the document names of a corpus. (Does not work for dfm objects, whose document names are fixed.)

Usage

```
docnames(x)

## S3 method for class 'corpus'
docnames(x)

docnames(x) <- value

## S3 method for class 'dfm'
docnames(x)
```

Arguments

x	the object with docnames
value	a character vector of the same length as x

Value

docnames returns a character vector of the document names

docnames<- assigns a character vector of the document names in a corpus

Examples

```
# query the document names of the inaugural speech corpus
docnames(inaugCorpus) <- paste("Speech", 1:ndoc(inaugCorpus), sep="")

# reassign the document names of the inaugural speech corpus
docnames(inaugCorpus) <- paste("Speech", 1:ndoc(inaugCorpus), sep="")
#
# query the document names of a dfm
docnames(dfm(inaugTexts[1:5]))
```

---

docvars *get or set for document-level variables*

Description

Get or set variables for the documents in a corpus

Usage

```
docvars(x, field = NULL)

docvars(x, field = NULL) <- value
```

Arguments

x	corpus whose document-level variables will be read or set
field	string containing the document-level variable name
value	the new values of the document-level variable

Value

docvars returns a data.frame of the document-level variables

docvars<- assigns value to the named field

Examples

```
head(docvars(inaugCorpus))
docvars(inaugCorpus, "President") <- paste("prez", 1:ndoc(inaugCorpus), sep="")
head(docvars(inaugCorpus))
```

---

encoding *get the encoding of documents in a corpus*

Description

Get or set the _encoding document-level metadata field in a corpus.

Usage

```
encoding(x, drop = TRUE)

encoding(x) <- value
```

Arguments

x	a corpus object
drop	return as a vector if TRUE, otherwise return a data.frame
value	a character vector or scalar representing the new value of the encoding (see Note)

Details

This function modifies the _encoding value set by metadoc. It is a wrapper for metadoc(corp, "encoding").

Note

This function differs from R's built-in Encoding function, which only allows the four values of "latin1", "UTF-8", "bytes", and "unknown" (and which assigns "unknown" to any text that contains only ASCII characters). Legal values for encodings must be from iconvlist. Note that encoding does not convert or set encodings, it simply records a user declaration of a valid encoding. (We hope to implement checking and conversion later.)

What if we replace all text sources with a “textsource(x, …)” function? A common interface but something much simpler than the madman tm approach.

`excel` *Function to declare a connection to an excel file*

Description

Function to declare a connection to a excel file.

Usage

```
excel(path = NULL, sheetIndex = 1)
```

Arguments

`path`	String describing the full path to the excel file or NULL to use a GUI to choose a directory from disk
`sheetIndex`	The index of the sheet of the excel file to read (as passed to read.xlsx2)

`features` *#compute the tf-idf weights of a dfm # #Returns a matrix of tf-idf weights, as a dfm object # #data(inaugCorpus) #dtm <- dfm(inaugCorpus) #dtm[1:10, 100:110] #tfidf(dtm)[1:10, 100:110] #tfidf(dtm, normalize=FALSE)[1:10, 100:110] #normalizes the term frequencies a dfm # #Returns a matrix of term weights, as a dfm object # #data(inaugCorpus) #dtm <- dfm(inaugCorpus) #dtm[1:10, 100:110] #tf(dtm)[1:10, 100:110] extract the feature labels from a dfm*

Description

Extract the features from a document-feature matrix, which are stored as the column names of the dfm object.

Usage

```
features(x)

## S3 method for class 'dfm'
features(x)
```

Arguments

`x`	the object (dfm) whose features will be extracted

Value

Character vector of the features

Examples

```
features(dfm(inaugTexts))[1:50]  # first 50 features (alphabetically sorted)
```

`flatten.dictionary` *Flatten a hierarchical dictionary into a list of character vectors*

Description

Converts a hierarchical dictionary (a named list of named lists, ending in character vectors at the lowest level) into a flat list of character vectors. Works like `unlist(dictionary, recursive=TRUE)` except that the recursion does not go to the bottom level.

Usage

```
flatten.dictionary(elms, parent = "", dict = list())
```

Arguments

`elms`	list to be flattened
`parent`	parent list name, gets built up through recursion in the same way that `unlist(dictionary, recursiv` works
`dict`	the bottom list of dictionary entries ("synonyms") passed up from recursive calls

Details

Called by dfm()

Value

A dictionary flattened down one level further than the one passed

Author(s)

Kohei Watanabe

Examples

```
dictPopulismEN <-
    list(populism=c("elit*", "consensus*", "undemocratic*", "referend*",
                    "corrupt*", "propagand", "politici*", "*deceit*",
                    "*deceiv*", "*betray*", "shame*", "scandal*", "truth*",
                    "dishonest*", "establishm*", "ruling*"))
flatten.dictionary(dictPopulismEN)

hdict <- list(level1a = list(level1a1 = c("l1a11", "l1a12"),
                             level1a2 = c("l1a21", "l1a22")),
              level1b = list(level1b1 = c("l1b11", "l1b12"),
                             level1b2 = c("l1b21", "l1b22", "l1b23")),
              level1c = list(level1c1a = list(level1c1a1 = c("lowest1", "lowest2")),
                             level1c1b = list(level1c1b1 = c("lowestalone"))))
flatten.dictionary(hdict)
```

getFBpage *Extract list of posts from a public Facebook page*

Description

getPage retrieves information from a public Facebook page. Note that information about users that have turned on the "follow" option on their profile can also be retrieved with this function. See Rfacebook package for additional methods to query the Facebook Graph API.

Usage

```
getFBpage(page, token, since = NULL, until = NULL, n = 100,
  feed = FALSE)
```

Arguments

page	A page ID or page name.
token	An access token created at https://developers.facebook.com/tools/explorer.
since	A UNIX timestamp or strtotime data value that points to the start of the time range to be searched. For more information on the accepted values, see: http://php.net/manual/en/function.strtotime.php
until	A UNIX timestamp or strtotime data value that points to the end of the time range to be searched. For more information on the accepted values, see: http://php.net/manual/en/function.strtotime.php
n	Number of posts of page to return. Note that number can be sometimes higher or lower, depending on status of API.
feed	If TRUE, the function will also return posts on the page that were made by others (not only the admin of the page).

Author(s)

Pablo Barbera

Examples

```
## Not run:
# scraping the 100 most recent posts on Barack Obama's page
token <- 'YOUR_FB_TOKEN_HERE'
pg <- getFBpage('barackobama', token, n=100)
# creating corpus object
fbcorpus <- corpus(pg)
summary(fbcorpus)
# viewing the DFM using a word cloud
fbDfm <- dfm(fbcorpus, stopwords=TRUE, stem=TRUE)
plot(fbDfm)

## End(Not run)
```

Internal only?

`getRootFileNames` *Truncate absolute filepaths to root filenames*

Description

This function takes an absolute filepath and returns just the document name

Usage

```
getRootFileNames(longFilenames)
```

Arguments

`longFilenames`	Absolute filenames including a full path with directory

Value

character vector of filenames withouth directory path

Author(s)

Paul Nulty

Examples

```
## Not run:
getRootFileNames('/home/paul/documents/libdem09.txt')

## End(Not run)
```

how is this different from directory()?

`getTextDir` *loads all text files from a given directory*

Description

given a directory name, get a list of all files in that directory and load them into a character vector using getTextFiles

Usage

```
getTextDir(dirname, pattern = "\\.txt$", enc = "unknown")
```

Arguments

`dirname`	A directory path
`pattern`	a regular expression pattern match for the input file names
`enc`	a value for encoding that is a legal value for Encoding

Value

character vector of texts read from disk

Author(s)

Paul Nulty

Examples

```
## Not run:
getTextDir('/home/paul/documents/')

## End(Not run)
```

---

getTextDirGui *provides a gui interface to choose a gui to load texts from*

Description

launches a GUI to allow the user to choose a directory from which to load all files.

Usage

```
getTextDirGui()
```

Value

character vector of texts read from disk

Author(s)

Paul Nulty

Examples

```
## Not run:
getTextFiles('/home/paul/documents/libdem09.txt')

## End(Not run)
```

---

getTextFiles *load text files from disk into a vector of character vectors points to files, reads them into a character vector of the texts with optional names, default being filenames returns a named vector of complete, unedited texts*

Description

load text files from disk into a vector of character vectors points to files, reads them into a character vector of the texts with optional names, default being filenames returns a named vector of complete, unedited texts

Usage

```
getTextFiles(filenames, textnames = NULL, enc = "unknown",
  verbose = FALSE)
```

Arguments

filenames	a vector of paths to text files
textnames	names to assign to the texts
enc	a value for encoding that is a legal value for Encoding
verbose	If TRUE, print out names of files being read. Default is FALSE

We need a consistent approach to encoding!

Value

character vector of texts read from disk

Author(s)

Paul Nulty

Examples

```
## Not run:
getTextFiles('/home/paul/documents/libdem09.txt')

## End(Not run)
```

---

getTimeline *return a time-line of most recent Tweets from a given user*

Description

Connect to the REST API of Twitter and returns up to 3,200 recent tweets sent by this user.

Usage

```
getTimeline(screen_name, numResults = 200, filename = "default", key,
  cons_secret, token, access_secret, df = TRUE)
```

Arguments

screen_name	user name of the Twitter user for which tweets will be downloaded
numResults	number of tweets to be downloaded (maximum is 3,200)
filename	file where tweets will be stored (in json format). If "default", they will be stored in a file whose name is the screen name of the queried user. If NA or NULL, tweets will be stored in a temporary file that will be deleted.
key	Key for twitter API authentication
cons_secret	for twitter API authentication
token	String for twitter API authentication
access_secret	for twitter API authentication
df	If TRUE, will return tweets in data frame format. If FALSE, will only store tweets in json format in disk.

Author(s)

Pablo Barbera

Examples

```
## Not run:
 key = 'your consumer key here'
 cons_secret = 'your consumer secret here'
 token = 'your access token here'
 access_secret = 'your access secret here'

 # download recent tweets by user "p_barbera"
 tweets <- getTimeline(screen_name="p_barbera", numResults=600,
     filename='p_barbera.json', key, cons_secret, token, access_secret)

 # creating corpus object
 twcorpus <- corpus(tweets)
 summary(twcorpus)

 # viewing the DFM using a word cloud
 twDfm <- dfm(twcorpus, stopwords=TRUE, stem=TRUE)
 plot(twDfm)

## End(Not run)
```

---

getTweets *Function to declare a twitter search*

Description

Function to declare a connection to a twitter search

Usage

```
getTweets(query, numResults = 50, key, cons_secret, token, access_secret)
```

Arguments

query	String describing the search query terms
numResults	Number of tweets to return. Maximum of approximately 1500
key	Key for twitter API authentication
cons_secret	for twitter API authentication
token	String for twitter API authentication
access_secret	for twitter API authentication

Value

The search results marked as a 'twitter' object for use by corpus.twitter()

inaugCorpus *A corpus of US presidential inaugural addresses from 1789-2013*

Description

inaugCorpus is the quanteda-package corpus object of US presidents' inaugural addresses since 1789. Document variables contain the year of the address and the last name of the president.

inaugTexts is the character vector of US presidential inaugaration speeches

References

https://archive.org/details/Inaugural-Address-Corpus-1789-2009 and http://www.presidency.ucsb.edu/inaugurals.php.

Examples

```
# some operations on the inaugural corpus
data(inaugCorpus)
summary(inaugCorpus)
head(docvars(inaugCorpus), 10)
# working with the character vector only
data(inaugTexts)
str(inaugTexts)
head(docvars(inaugCorpus), 10)
mycorpus <- corpus(inaugTexts)
```

json *Function to read files with tweets in JSON format*

Need to associate more clearly with tweets - if this is not general then it should not be called just json

Description

Function to read files with tweets in JSON format

Usage

```
json(path = NULL, source = "twitter", enc = "unknown", ...)
```

Arguments

path	string describing the full path of a directory that contains files in json format, or a vector of file names in in json format
source	source of data in JSON format.
enc	encoding of the input json file
...	additional arguments passed to parseTweets

Examples

```
## Not run:
# name a directory of files in json format
tweets <- json("~/Dropbox/QUANTESS/corpora/tweets")
corpus(tweets)

# read a single file in json format
tweets <- json("~/Dropbox/QUANTESS/corpora/tweets/BarackObama.json")
corpus(tweets)

## End(Not run)
```

---

kwic *List key words in context from a text or a corpus of texts.*

Description

For a text or a collection of texts (in a quanteda corpus object), return a list of a keyword supplied by the user in its immediate context, identifying the source text and the word index number within the source text. (Not the line number, since the text may or may not be segmented using end-of-line delimiters.)

Usage

```
kwic(x, word, window = 5, regex = TRUE)

## S3 method for class 'character'
kwic(x, word, window = 5, regex = TRUE)

## S3 method for class 'corpus'
kwic(x, word, window = 5, regex = TRUE)
```

Arguments

x	A text character scalar or a quanteda corpus. (Currently does not support character vectors.)
word	A keyword chosen by the user.
window	The number of context words to be displayed around the keyword.
regex	If TRUE (default), then "word" is a regular expression, otherwise only match the whole word. Note that if regex=TRUE and no special regular expression characters are used in the search query, then the concordance will include all words in which the search term appears, and not just when it appears as an entire word. (For instance, searching for the word "key" will also return "whiskey".)

Value

A data frame with the context before (preword), the keyword in its original format (word, preserving case and attached punctuation), and the context after (postword). The rows of the dataframe will be named with the word index position, or the text name and the index position for a corpus object.

Author(s)

Kenneth Benoit and Paul Nulty

Examples

```
kwic(inaugTexts, "terror")
kwic(inaugTexts, "terror", regex=FALSE)  # returns only whole word, without trailing punctuation
```

---

language *get or set the language of corpus documents*

Description

Get or set the _language document-level metadata field in a corpus.

Usage

```
language(corp, drop = TRUE)

language(corp) <- value
```

Arguments

corp	a corpus object
drop	return as a vector if TRUE, otherwise return a data.frame
value	the new value for the language meta-data field, a string or character vector equal in length to ndoc(corp)

Details

This function modifies the _language value set by metadoc. It is a wrapper for metadoc(corp, "language").

MCMCirtPoisson1d *Bayesian-MCMC version of a 1-dimensional Poisson IRT scaling model*

Description

MCMCirtPoisson1d implements a flexible, Bayesian model estimated in JAGS using MCMC. It is based on the implementation of wordfish from the austin package. Options include specifying a model for alpha using document-level covariates, and partitioning the word parameters into different subsets, for instance, countries.

Usage

```
MCMCirtPoisson1d(dtm, dir = c(1, 2), control = list(sigma = 3, startparams =
  NULL), verbose = TRUE, itembase = 0, startRandom = FALSE, nChains = 1,
  nAdapt = 100, nUpdate = 300, nSamples = 200, nThin = 1, ...)
```

Arguments

dtm	The document-term matrix. Ideally, documents form the rows of this matrix and words the columns, although it should be correctly coerced into the correct shape.
dir	A two-element vector, enforcing direction constraints on theta and beta, which ensure that `theta[dir[1]] < theta[dir[2]]`. The elements of `dir` will index documents.
control	list specifies options for the estimation process. These are: `tol`, the proportional change in log likelihood sufficient to halt estimation, `sigma` the standard deviation for the beta prior in poisson form, and `startparams` a previously fitted wordfish model. `verbose` generates a running commentary during estimation. See `wordfish`.
verbose	Turn this on for messages. Default is `TRUE`.
itembase	Item constraints for identifying the model. Options are:
	`0` (default) Use the sum to zero constraint, on the item parameters, such that $\sum_j \psi_j = \sum_j \beta_j = 0$.
	integer or feature label A index or column name from the input dfm indicating which item should be used as the reference category, for setting corner constrants on the item paramters such that $\psi_j = \beta_j = 0$.
	NULL Do not use item constraints, and hope that the mode is identified by setting $\theta_i \sim N(0, 1)$.
startRandom	`FALSE` by default, uses random starting values (good for multiple chains) if `TRUE`
nChains	Number of chains to run in JAGS.
nAdapt	Adaptation iterations in JAGS.
nUpdate	Update iterations in JAGS.
nSamples	Number of posterior samples to draw in JAGS.
nThin	Thinning parameter for drawing posterior samples in JAGS.
...	Additional arguments passed through to MCMCirtPoisson1d and to JAGS.

Details

`textmodel_wordfish`

The ability to constrain an item is designed to make the additive Poisson GLM mathematically equivalent to the multinomial model for $R \times C$ contingency tables. We recommend using the default setting of `itembase=0`, or setting a "neutral" feature to have $\psi_0 = 0 and \beta_0 = 0$, for example the word "the" for a text count model (assuming this word has not been removed). Note: Currently the item-level return values will be returned in the original order suppled (`psi` and `beta`) but this is not true yet for the `mcmc.samples` value, which will have the constrained category as index 1. (We will fix this soon.)

Value

An augmented `wordfish` class object with additional stuff packed in. To be documented.

Author(s)

Kenneth Benoit

Examples

```
## Not run:
library(quantedaData)
data(ie2010Corpus)
ieDfm <- dfm(ie2010Corpus)
# estimate the maximium likelihood wordfish model from austin
wf <- textmodel_wordfish(ieDfm, dir=c(2,1))

# estimate the MCMC model, default values
wfMCMCstz   <- textmodel_wordfish(ieDfm, method="mcmc", dir=c(2,1))
wfMCMCthe   <- textmodel_wordfish(ieDfm, method="mcmc", itembase="the", dir=c(2,1))
wfMCMCuncon <- textmodel_wordfish(ieDfm, method="mcmc", itembase=NULL, dir=c(2,1))

# compare the estimates of \theta_i
require(psych)
pairs.panels(data.frame(ML=wf$theta,
                        MCMCbase=wfMCMC$theta,
                        MCMCuncon=wfMCMCuncon$theta),
             smooth=FALSE, scale=FALSE, ellipses=FALSE, lm=TRUE, cex.cor=2.5)
# inspect a known "opposition" word beta values
wfMCMCstz$beta[which(wfMCMCstz$words=="fianna")]
wfMCMCthe$beta[which(wfMCMCstz$words=="fianna")]
wfMCMCuncon$beta[which(wfMCMCthe$words=="fianna")]

# random starting values, for three chains
dtm.sample <- trim(dtm, sample=200)
wfMCMCsample <- MCMCirtPoisson1d(dtm.sample, dir=c(2,1), startRandom=TRUE, nChains=3)

## End(Not run)
```

`metacorpus` *get or set corpus metadata*

Description

Get or set the corpus-level metadata in a quanteda corpus object.

Usage

```
metacorpus(corp, field = NULL)

metacorpus(corp, field) <- value
```

Arguments

`corp`	A quanteda corpus object
`field`	Metadata field name(s). If `NULL` (default), return all metadata names.
`value`	new value of the corpus metadata field

Value

For `metacorpus`, a list of the metadata fields in the corpus. If a list is not what you wanted, you can wrap the results in unlist, but this will remove any metadata field that is set to `NULL`.

For `metacorpus <-`, the corpus with the updated metadata.

Examples

```
metacorpus(inaugCorpus)
metacorpus(inaugCorpus, "source")
metacorpus(inaugCorpus, "citation") <- "Presidential Speeches Online Project (2014)."
metacorpus(inaugCorpus, "citation")
```

---

metadoc *get or set document-level meta-data*

Description

Get or set the document-level meta-data, including reserved fields for language and corpus.

Usage

```
metadoc(corp, field = NULL)

metadoc(corp, field = NULL) <- value
```

Arguments

corp	A quanteda corpus object
field	string containing the name of the metadata field(s) to be queried or set
value	the new value of the new meta-data field

Value

For `texts`, a character vector of the texts in the corpus.

For `texts <-`, the corpus with the updated texts.

Note

Document-level meta-data names are preceded by an underscore character, such as `_encoding`, but when named in in the `field` argument, do *not* need the underscore character.

Examples

```
mycorp <- subset(inaugCorpus, Year>1990)
summary(mycorp, showmeta=TRUE)
metadoc(mycorp, "encoding") <- "UTF-8"
metadoc(mycorp)
metadoc(mycorp, "language") <- "english"
summary(mycorp, showmeta=TRUE)
```

ndoc *get the number of documents or features*

Description

Returns the number of documents or features in a quanteda object.

Usage

```
ndoc(x)

## S3 method for class 'corpus'
ndoc(x)

## S3 method for class 'dfm'
ndoc(x, ...)

nfeature(x)

## S3 method for class 'corpus'
nfeature(x)

## S3 method for class 'dfm'
nfeature(x)
```

Arguments

x	a corpus or dfm object
...	additional parameters

Value

an integer (count) of the number of documents or features in the corpus or dfm

Examples

```
ndoc(inaugCorpus)
ndoc(dfm(inaugCorpus))
nfeature(dfm(inaugCorpus))
nfeature(trimdfm(dfm(inaugCorpus), minDoc=5, minCount=10))
```

ngrams *Create ngrams*

Description

Create a set of ngrams (words in sequence) from text(s) in a character vector

Usage

```
ngrams(text, n = 2, concatenator = "_", include.all = FALSE, ...)
```

Arguments

text	character vector containing the texts from which ngrams will be extracted
n	the number of tokens to concatenate. Default is 2 for bigrams.
concatenator	character for combining words, default is _ (underscore) character
include.all	if TRUE, add n-1...1 grams to the returned list
...	additional parameters

Details

... provides additional arguments passed to tokenize

Value

a list of character vectors of ngrams, one list element per text

Author(s)

Ken Benoit, Kohei Watanabe, Paul Nulty

Examples

```
ngrams("The quick brown fox jumped over the lazy dog.", n=2)
identical(ngrams("The quick brown fox jumped over the lazy dog.", n=2),
          bigrams("The quick brown fox jumped over the lazy dog.", n=2))
ngrams("The quick brown fox jumped over the lazy dog.", n=3)
ngrams("The quick brown fox jumped over the lazy dog.", n=3, concatenator="~")
ngrams("The quick brown fox jumped over the lazy dog.", n=3, include.all=TRUE)
```

---

nresample *get the number of resamples*

Description

Get the number of resamples from a corpus or dfm object

Usage

```
nresample(x)

## S3 method for class 'corpus'
nresample(x)

## S3 method for class 'dfm'
nresample(x)
```

Arguments

x	corpus object containing the texts to be resampled

Value

an integer as the number of resampled texts

---

plot.dfm *plot features as a wordcloud*

Description

The default plot method for a dfm object. Produces a wordcloud plot for the features of the dfm, weighted by the total frequencies. To produce word cloud plots for specific documents, the only way currently to do this is to produce a dfm only from the documents whose features you want plotted.

Usage

```
## S3 method for class 'dfm'
plot(x, ...)
```

Arguments

x	a dfm object
...	additional parameters passed to to wordcloud or to text (and strheight, strwidth)

See Also

wordcloud

Examples

```
# plot the features (without stopwords) from Obama's two inaugural addresses
mydfm <- dfm(subset(inaugCorpus, President=="Obama"), verbose=FALSE, stopwords=TRUE)
plot(mydfm)

# plot only Lincoln's inaugural address
plot(dfm(subset(inaugCorpus, President=="Lincoln"), verbose=FALSE, stopwords=TRUE))

# plot in colors with some additional options passed to wordcloud
plot(mydfm, random.color=TRUE, rot.per=.25, colors=sample(colors()[2:128], 5))
```

`predict.textmodel` *predict a text model on new data*

Description

Apply a fitted text model to make predictions on test data.

implements class predictions using trained Naive Bayes examples

Usage

```
## S3 method for class 'textmodel'
predict(object, ...)

## S3 method for class 'textmodelfitted'
predict(object, ...)

## S3 method for class 'naivebayes'
predict(object, newdata = NULL, scores = c(-1, 1), ...)

## S3 method for class 'wordscores'
predict(object, newdata = NULL, rescaling = "none",
  level = 0.95, verbose = TRUE, ...)
```

Arguments

`object`	a fitted textmodel object (from `textmodel`)
`...`	further arguments passed to or from other methods
`newdata`	A dfm or matrix object containing features found in the training object. If omitted, the original dfm on which the model was fit will be used.
`scores`	"reference" values when the wordscores equivalent implementation of Naive Bayes prediction is used. Default is `c(-1, 1)`.
`rescaling`	`none` for "raw" scores; `lbg` for LBG (2003) rescaling; or `mv` for the rescaling proposed by Martin and Vanberg (2007). (Note to authors: Provide full details here in documentation.)
`level`	probability level for confidence interval width
`verbose`	output status messages if `TRUE`

Value

A list of two data frames, named `docs` and `words` corresponding to word- and document-level predicted quantities

`docs`	data frame with document-level predictive quantities: nb.predicted, ws.predicted, bs.predicted, PcGw, wordscore.doc, bayesscore.doc, posterior.diff, posterior.logdiff. Note that the diff quantities are currently implemented only for two-class solutions.
`words`	data-frame with word-level predictive quantities: wordscore.word, bayesscore.word

predict.wordscores returns a data.frame whose rows are the documents fitted and whose columns contain the scored textvalues, with the number of columns depending on the options called (for instance, how many rescaled scores, and whether standard errors were requested.) (Note: We may very well change this soon so that it is a list similar to other existing fitted objects.)

Author(s)

Kenneth Benoit

Ken Benoit, borrowed in places from Will Lowe, who probably borrowed from me at some early stage.

References

LBG (2003); Martin and Vanberg (2007)

---

print,dfmSparse-method *print a dfm object*

Description

print method for dfm objects

Usage

```
## S4 method for signature 'dfmSparse'
print(x, show.values = FALSE, show.settings = FALSE,
  ...)

## S4 method for signature 'dfmDense'
print(x, show.values = FALSE, show.settings = FALSE,
  ...)

## S3 method for class 'dfm'
print(x, show.values = FALSE, show.settings = FALSE, ...)
```

Arguments

x	the dfm to be printed
show.values	print the dfm as a matrix or array (if resampled).
show.settings	Print the settings used to create the dfm. See settings.
...	further arguments passed to or from other methods

print.textmodel *print a textmodel object*

Description

print a class of textmodel object

Usage

```
## S3 method for class 'textmodel'
print(x, ...)

## S3 method for class 'textmodelfitted'
print(x, ...)

## S3 method for class 'textmodelpredicted'
print(x, ...)
```

Arguments

x	textmodel object to be printed
...	additional arguments passed to print

readLIWCdict *Import a LIWC-formatted dictionary*

need a general function for dictionary import, format=c("LIWC", "wordstat", etc.)

Description

Make a flattened dictionary list object from a LIWC dictionary file.

Usage

```
readLIWCdict(path, maxcats = 10, enc = "")
```

Arguments

path	full pathname of the LIWC-formatted dictionary file (usually a file ending in .dic)
maxcats	the maximum number of categories to read in, set by the maximum number of dictionary categories that a term could belong to. For non-exclusive schemes such as the LIWC, this can be up to 7. Set to 10 by default, which ought to be more than enough.
enc	a valid input encoding for the file to be read, see iconvlist

Value

a dictionary class named list, where each the name of element is a bottom level category in the hierarchical wordstat dictionary. Each element is a list of the dictionary terms corresponding to that level.

Author(s)

Kenneth Benoit

Examples

```
## Not run:
LIWCdict <- readLIWCdict("~/Dropbox/QUANTESS/corpora/LIWC/LIWC2001_English.dic")
## End(Not run)
```

`readWStatDict` *Import a Wordstat dictionary*

Description

Make a flattened list from a hierarchical wordstat dictionary

Usage

```
readWStatDict(path, enc = "", lower = TRUE)
```

Arguments

`path`	full pathname of the wordstat dictionary file (usually ending in .cat)
`enc`	a valid input encoding for the file to be read, see iconvlist
`lower`	if `TRUE` (default), convert the dictionary entries to lower case

Value

a named list, where each the name of element is a bottom level category in the hierarchical wordstat dictionary. Each element is a list of the dictionary terms corresponding to that level.

Author(s)

Kohei Watanabe, Kenneth Benoit

Examples

```
## Not run:
path <- '~/Dropbox/QUANTESS/corpora/LaverGarry.cat'
lgdict <- readWStatDict(path)

## End(Not run)
```

resample *resampling methods for a corpus*

Description

Draw a set of random resamples from a corpus object, at a specified level of resampling, and record additional "resampled texts" as document-level metadata, stored as `_resampleXXX` for the XXXth resample.

Usage

```
resample(x, ...)

## S3 method for class 'corpus'
resample(x, n = 100, unit = c("sentences", "paragraphs"),
  ...)

is.resampled(x)

## S3 method for class 'corpus'
is.resampled(x)

## S3 method for class 'dfm'
is.resampled(x)
```

Arguments

`x`	corpus object containing the texts to be resampled
`...`	additional arguments passed to `segment`
`n`	number of resamples to be drawn
`unit`	resampling unit for drawing the random samples, can be `sentences` or `paragraphs`.

Details

`is.resampled` checks a corpus or dfm object and returns `TRUE` if these contain resampled texts or the results of resampled texts

Value

a corpus object containing new resampled texts.

Note

Additional resampling units to be added will include fixed length samples and random length samples.

Examples

```
testCorp <- resample(subset(inaugCorpus, Year>2000), 10, "sentences")
testCorpPara <- resample(corpus(uk2010immig), 10, "paragraphs")
names(metadoc(testCorp))
x <- corpus(c("Sentence One C1.  Sentence Two C1.  Sentence Three C1.",
              "Sentence One C2.  Sentence Two C2.  Sentence Three C2.
               Sentence Four C2.  Sentence Five C2.  Sentence Six C2."),
            docnames=c("docTwo", "docOne"))
testRS <- resample(x, n=3)
metadoc(testRS)
testRS$documents

# tests to see if a corpus contains resampled texts
is.resampled(testCorp)
is.resampled(inaugCorpus)
```

see also nsample

`segmentSentence` *segment texts into component elements*

Description

Segment text(s) into tokens, sentences, paragraphs, or other sections. `segment` works on a character vector or corpus object, and allows the delimiters to be defined. See details.

Usage

```
~~segmentSentence~~(x, delimiter = "[.!?:;]")

~~segmentParagraph~~(x, delimiter = "\\n{2}")

segment(x, ...)

## S3 method for class 'character'
segment(x, what = c("tokens", "sentences", "paragraphs",
  "tags", "other"), delimiter = ifelse(what == "tokens", " ", ifelse(what ==
  "sentences", "[.!?:;]", ifelse(what == "paragraphs", "\\n{2}", ifelse(what
  == "tags", "##\\w+\\b", NULL)))), ...)

## S3 method for class 'corpus'
segment(x, what = c("tokens", "sentences", "paragraphs",
  "tags", "other"), delimiter = ifelse(what == "tokens", " ", ifelse(what ==
  "sentences", "[.!?:;]", ifelse(what == "paragraphs", "\\n{2}", ifelse(what
  == "tags", "##\\w+\\b", NULL)))), ...)
```

Arguments

`x`	text or corpus object to be segmented
`delimiter`	delimiter defined as a regex for segmentation. Each type has its own default, except `other`, which requires a value to be specified.
`...`	provides additional arguments to be passed to clean

detail these; defaults are near impossible to read in the function formals

what — unit of segmentation. Current options are tokens, sentences, paragraphs, and other. Segmenting on other allows segmentation of a text on any user-defined value, and must be accompanied by the delimiter argument.

Details

Tokens are delimited by whitespace. For sentences, the delimiter can be defined by the user. The default for sentences includes ., !, ?, plus ; and :.

For paragraphs, the default is two carriage returns, although this could be changed to a single carriage return by changing the value of delimiter to "\\n{1}" which is the R version of the regex for one newline character. (You might need this if the document was created in a word processor, for instance, and the lines were wrapped in the window rather than being hard-wrapped with a newline character.)

Value

segmentSentence returns a character vector of sentences that have been segmented

segmentParagraph returns a character vector of paragraphs that have been segmented

A list of segmented texts, with each element of the list correponding to one of the original texts.

Note

Does not currently record document segments if segmenting a multi-text corpus into smaller units. For this, use changeunits instead.

Examples

```
# segment sentences of the UK 2010 immigration sections of manifestos
segmentSentence(uk2010immig[1])[1:5]   # 1st 5 sentences from first (BNP) text
str(segmentSentence(uk2010immig[1]))   # a 132-element char vector
str(segmentSentence(uk2010immig[1:2])) # a 144-element char vector (143+ 12)
# segment paragraphs
segmentParagraph(uk2010immig[3])[1:2]   # 1st 2 Paragraphs from 3rd (Con) text
str(segmentParagraph(uk2010immig[3]))   # a 12-element char vector
# same as tokenize()
identical(tokenize(uk2010immig, lower=FALSE), segment(uk2010immig, lower=FALSE))

# segment into paragraphs
segment(uk2010immig[3:4], "paragraphs")

# segment a text into sentences
segmentedChar <- segment(uk2010immig, "sentences")
segmentedChar[2]
testCorpus <- corpus("##INTRO This is the introduction.
                     ##DOC1 This is the first document.
                     Second sentence in Doc 1.
                     ##DOC3 Third document starts here.
                     End of third document.")
testCorpusSeg <- segment(testCorpus, "tags")
summary(testCorpusSeg)
texts(testCorpusSeg)
# segment a corpus into sentences
segmentedCorpus <- segment(corpus(uk2010immig), "sentences")
identical(segmentedCorpus, segmentedChar)
```

`settings` *Get or set the corpus settings*

Description

Get or set the corpus settings

Get or set various settings in the corpus for the treatment of texts, such as rules for stemming, stopwords, collocations, etc.

Get the settings from a which a dfm was created

Needs tidying up and elaboration here.

Usage

```
settings(x, ...)

## S3 method for class 'corpus'
settings(x, field = NULL, ...)

settings(x, field) <- value

## S3 method for class 'dfm'
settings(x, ...)

print.settings(x, ...)
```

Arguments

`x`	object from/to which settings are queried or applied
`...`	additional arguments
`field`	string containing the name of the setting to be set or queried `settings(x)` query the corps settings `settings(x, field) <-` update the corpus settings for `field`
`value`	new setting value

Examples

```
settings(inaugCorpus, "stopwords")
tempdfm <- dfm(inaugCorpus)
tempdfmSW <- dfm(inaugCorpus, stopwords=TRUE)
settings(inaugCorpus, "stopwords") <- TRUE
tempdfmSW <- dfm(inaugCorpus)
tempdfm <- dfm(inaugCorpus, stem=TRUE)
settings(tempdfm)
```

`settingsInitialize` `settingsInitialize` *returns a list of legal settings, set to their default values*

Description

`settingsInitialize` returns a list of legal settings, set to their default values

Usage

Needs rethinking…

```
settingsInitialize()
```

`similarity` *compute similarities between documents and/or features*

Description

Compute similarities between documents and/or features from a `dfm`. Uses the similarity measures defined in simil. See `pr_DB` for available distance measures, or how to create your own.

Usage

```
similarity(x, selection, n = 10, margin = c("features", "documents"),
  method = "correlation", sort = TRUE, normalize = TRUE, digits = 4)

## S4 method for signature 'dfm,index'
similarity(x, selection, n = 10,
  margin = c("features", "documents"), method = "correlation",
  sort = TRUE, normalize = TRUE, digits = 4)
```

Arguments

`x`	a dfm object
`selection`	character or character vector of document names or feature labels from the dfm
`n`	the top `n` most similar items will be returned, sorted in descending order. If `n` is `NULL`, return all items.
`margin`	identifies the margin of the dfm on which similarity will be computed: `features` for word/term features or `documents` for documents.
`method`	a valid method for computing similarity from `pr_DB`
`sort`	sort results in descending order if `TRUE`
`normalize`	if `TRUE`, normalize the dfm by term frequency within document (so that the dfm values will be relative term frequency within each document)
`digits`	digits for rounding results

Value

a named list of the selection labels, with a sorted named vector of similarity measures.

Note

The method for computing feature similarities can be quite slow when there are large numbers of feature types. Future implementations will hopefully speed this up.

Examples

```
# create a dfm from inaugural addresses from Reagan onwards
presDfm <- dfm(subset(inaugCorpus, Year>1980), stopwords=TRUE, stem=TRUE)

# compute some document similarities
similarity(presDfm, "1985-Reagan", n=5, margin="documents")
similarity(presDfm, c("2009-Obama" , "2013-Obama"), n=5, margin="documents")
similarity(presDfm, c("2009-Obama" , "2013-Obama"), n=NULL, margin="documents")
similarity(presDfm, c("2009-Obama" , "2013-Obama"), n=NULL, margin="documents", method="cosine")
similarity(presDfm, "2005-Bush", n=NULL, margin="documents", method="eJaccard", sort=FALSE)

# compute some term similarities
similarity(presDfm, c("fair", "health", "terror"), method="cosine")

## Not run:

# compare to tm
require(tm)
data("crude")
crude <- tm_map(crude, content_transformer(tolower))
crude <- tm_map(crude, removePunctuation)
crude <- tm_map(crude, removeNumbers)
crude <- tm_map(crude, stemDocument)
tdm <- TermDocumentMatrix(crude)
findAssocs(tdm, c("oil", "opec", "xyz"), c(0.75, 0.82, 0.1))
# in quanteda
crudeDfm <- dfm(corpus(crude))
similarity(crudeDfm, c("oil", "opec", "xyz"), normalize=FALSE, digits=2)

## End(Not run)
```

`sort.dfm` *sort a dfm by one or more margins*

Description

Sorts a dfm by frequency of total features, total features in documents, or both

Usage

by = c("frequency", "labels")

```
## S3 method for class 'dfm'
sort(x, decreasing = TRUE, margin = c("features", "docs",
  "both"), ...)
```

Arguments

x	Document-feature matrix created by dfm
decreasing	TRUE (default) if sort will be in descending order, otherwise sort in increasing order
margin	which margin to sort on features to sort by frequency of features, docs to sort by total feature counts in documents, and both to sort by both
...	additional arguments passed to base method sort.int

Value

A sorted dfm matrix object

Author(s)

Ken Benoit

Examples

```
dtm <- dfm(inaugCorpus)
dtm[1:10, 1:5]
dtm <- sort(dtm)
sort(dtm)[1:10, 1:5]
sort(dtm, TRUE, "both")[1:10, 1:5]  # note that the decreasing=TRUE argument
                                    # must be second, because of the order of the
                                    # formals in the generic method of sort()
```

---

statLexdiv *calculate lexical diversity*

Description

Calculate the lexical diversity or complexity of text(s).

Usage

```
statLexdiv(x, ...)

## S3 method for class 'dfm'
statLexdiv(x, measure = c("TTR", "C", "R", "CTTR", "U", "S",
  "Maas"), log.base = 10, ...)

## S3 method for class 'numeric'
statLexdiv(x, measure = c("TTR", "C", "R", "CTTR", "U", "S",
  "Maas"), log.base = 10, ...)
```

Arguments

x	a document-feature matrix object
...	additional arguments
measure	A character vector defining the measure to calculate.
log.base	A numeric value defining the base of the logarithm.

Details

`statLexdiv` calculates a variety of proposed indices for lexical diversity. In the following formulae, N refers to the total number of tokens, and V to the number of types:

`"TTR"`: The ordinary *Type-Token Ratio*:

$$TTR = \frac{V}{N}$$

`"C"`: Herdan's *C* (Herdan, 1960, as cited in Tweedie & Baayen, 1998; sometimes referred to as *LogTTR*):

$$C = \frac{\lg V}{\lg N}$$

`"R"`: Guiraud's *Root TTR* (Guiraud, 1954, as cited in Tweedie & Baayen, 1998):

$$R = \frac{V}{\sqrt{N}}$$

`"CTTR"`: Carroll's *Corrected TTR*:

$$CTTR = \frac{V}{\sqrt{2N}}$$

`"U"`: Dugast's *Uber Index* (Dugast, 1978, as cited in Tweedie & Baayen, 1998):

$$U = \frac{(\lg N)^2}{\lg N - \lg V}$$

`"S"`: Summer's index:

$$S = \frac{\lg \lg V}{\lg \lg N}$$

`"K"`: Yule's *K* (Yule, 1944, as cited in Tweedie & Baayen, 1998) is calculated by:

$$K = 10^4 \times \frac{(\sum_{X=1}^{X} f_X X^2) - N}{N^2}$$

where N is the number of tokens, X is a vector with the frequencies of each type, and f_X is the frequencies for each X.

`"Maas"`: Maas' indices (a, $\lg V_0$ & $\lg {}_e V_0$):

$$a^2 = \frac{\lg N - \lg V}{\lg N^2}$$

$$\lg V_0 = \frac{\lg V}{\sqrt{1 - \frac{\lg V}{\lg N}^2}}$$

The measure was derived from a formula by M\"uller (1969, as cited in Maas, 1972). $\lg {}_e V_0$ is equivalent to $\lg V_0$, only with e as the base for the logarithms. Also calculated are a, $\lg V_0$ (both not the same as before) and V' as measures of relative vocabulary growth while the text progresses. To calculate these measures, the first half of the text and the full text will be examined (see Maas, 1972, p. 67 ff. for details). Note: for the current method (for a dfm) there is no computation on separate halves of the text.

Value

a vector of lexical diversity statistics, each corresponding to an input document

Note

This implements only the static measures of lexical diversity, not more complex measures based on windows of text such as the Mean Segmental Type-Token Ratio, the Moving-Average Type-Token Ratio (Covington & McFall, 2010), the MLTD or MLTD-MA (Moving-Average Measure of Textual Lexical Diversity) proposed by McCarthy & Jarvis (2010) or Jarvis (no year), or the HD-D version of vocd-D (see McCarthy & Jarvis, 2007). These are available from the package **korRpus**.

Author(s)

Kenneth Benoit, adapted from the S4 class implementation written by Meik Michalke in the **koRpus** package.

References

Covington, M.A. & McFall, J.D. (2010). Cutting the Gordian Knot: The Moving-Average Type-Token Ratio (MATTR). *Journal of Quantitative Linguistics*, 17(2), 94–100.

Maas, H.-D., (1972). \"Uber den Zusammenhang zwischen Wortschatzumfang und L\"ange eines Textes. *Zeitschrift f\"ur Literaturwissenschaft und Linguistik*, 2(8), 73–96.

McCarthy, P.M. & Jarvis, S. (2007). vocd: A theoretical and empirical evaluation. *Language Testing*, 24(4), 459–488.

McCarthy, P.M. & Jarvis, S. (2010). MTLD, vocd-D, and HD-D: A validation study of sophisticated approaces to lexical diversity assessment. *Behaviour Research Methods*, 42(2), 381–392.

Michalke, Meik. (2014) *koRpus: An R Package for Text Analysis*. Version 0.05-5. http://reaktanz.de/?c=hacking&s=koRpus

Tweedie. F.J. & Baayen, R.H. (1998). How Variable May a Constant Be? Measures of Lexical Richness in Perspective. *Computers and the Humanities*, 32(5), 323–352.

Examples

```
mydfm <- dfm(inaugCorpus)
mydfmSW <- dfm(inaugCorpus, stopwords=TRUE)
results <- data.frame(TTR = statLexdiv(mydfm, "TTR"),
                      CTTR = statLexdiv(mydfm, "CTTR"),
                      U = statLexdiv(mydfm, "U"),
                      TTRs = statLexdiv(mydfmSW, "TTR"),
                      CTTRs = statLexdiv(mydfmSW, "CTTR"),
                      Us = statLexdiv(mydfmSW, "U"))
results
cor(results)
t(statLexdiv(mydfmSW, "Maas"))
```

`stopwords` *A named list containing common stopwords in 14 languages*

---

change to .stopwords?

Description

SMART English stopwords from the SMART information retrieval system (obtained from http://jmlr.csail.mit.edu/papers/smart-stop-list/english.stop) and a set of stopword lists from the Snowball stemmer project in different languages (obtained from http://svn.tartarus.org/snowball/trunk/website/algorithms/*/stop.txt).

Supported languages are danish, dutch, english, finnish, french, german, hungarian, italian, norwegian, portuguese, russian, spanish, and swedish. Language names are case sensitive. Alternatively, their IETF language tags may be used.

Also now (>=0.6.3) includes Arabic stopwords.

`stopwordsGet` *access stopwords* just stopwords("")? or stopwords("all") for the list?

Description

This function retrieves stopwords from the type specified in the `kind` argument and returns the stopword list as a character vector The default is English. See stopwords for information about the list.

Usage

```
stopwordsGet(kind = "english")
```

Arguments

`kind`	The pre-set kind of stopwords (as a character string)

Value

a character vector or dfm with stopwords removed

Examples

```
stopwordsGet()
stopwordsGet("italian")
```

`stopwordsRemove` *remove stopwords from a text or dfm* ignoreFeatures()

Description

This function takes a character vector or dfm and removes words in the remove common or 'semantically empty' words from a text. See stopwordsGet for the information about the default lists.

Usage

```
stopwordsRemove(x, stopwords = NULL, verbose = TRUE)

## S3 method for class 'character'
stopwordsRemove(x, stopwords = NULL, verbose = TRUE)

## S3 method for class 'dfm'
stopwordsRemove(x, stopwords = NULL, verbose = TRUE)

## S3 method for class 'dfmSnum'
```

```
stopwordsRemove(x, stopwords = NULL, verbose = TRUE)

## S3 method for class 'collocations'
stopwordsRemove(x, stopwords = NULL, verbose = TRUE)
```

Arguments

x	Object from which stopwords will be removed
stopwords	Character vector of stopwords to remove. Now requires an explicit list to be supplied, for instance `stopwordsGet("english")`.
verbose	if TRUE print message about how many items were removed

Details

This function takes a character vector 'text' and removes words in the list provided in stopwords. If no list of stopwords is provided a default list for English is used. The function stopwordsGet can load a default set of stopwords for many languages.

Value

an object with stopwords removed

Examples

```
## examples for character objects
someText <- "Here's some text containing words we want to remove."
stopwordsRemove(someText, stopwordsGet("english"))
stopwordsRemove(someText, stopwordsGet("SMART"))
stopwordsRemove(someText, c("some", "want"))
itText <- "Ecco alcuni di testo contenente le parole che vogliamo rimuovere."
stopwordsRemove(itText, stopwordsGet("italian"))

## example for dfm objects
mydfm <- dfm(uk2010immig, verbose=FALSE)
stopwordsRemove(mydfm, stopwordsGet("english"))
mydfms <- dfm(uk2010immig, verbose=FALSE, matrixType="sparse")
dim(stopwordsRemove(mydfms, stopwordsGet("SMART")))

## example for collocations
(myCollocs <- collocations(inaugTexts, top=20))
stopwordsRemove(myCollocs, stopwordsGet("english"))
```

---

subset.corpus *extract a subset of a corpus*

Description

eventually: we could implement some dplyr methods

Works just like the normal subset command but for corpus objects

Usage

```
## S3 method for class 'corpus'
subset(x, subset = NULL, select = NULL, ...)
```

Arguments

x	corpus object to be subsetted.
subset	logical expression indicating elements or rows to keep: missing values are taken as false.
select	expression, indicating the attributes to select from the corpus
...	additional arguments affecting the summary produced

Value

corpus object

Examples

```
summary(subset(inaugCorpus, Year>1980))
summary(subset(inaugCorpus, Year>1930 & President=="Roosevelt", select=Year))
```

---

summary.corpus *Corpus summary*

Description

Displays information about a corpus object, including attributes and metadata such as date of number of texts, creation and source.

Usage

```
## S3 method for class 'corpus'
summary(object, n = 100, verbose = TRUE,
  showmeta = FALSE, ...)
```

Arguments

object	corpus to be summarized
n	maximum number of texts to describe, default=100
verbose	FALSE to turn off printed output
showmeta	TRUE to include document-level meta-data
...	additional arguments affecting the summary produced

Examples

```
summary(inaugCorpus)
summary(inaugCorpus, n=10)
mycorpus <- corpus(uk2010immig, docvars=data.frame(party=names(uk2010immig)), enc="UTF-8")
summary(mycorpus, showmeta=TRUE)  # show the meta-data
mysummary <- summary(mycorpus, verbose=FALSE)  # (quietly) assign the results
mysummary$Types / mysummary$Tokens             # crude type-token ratio
```

summary.textmodel *summary of a textmodel object*

Description

Summarize the results of a fitted or predicted textmodel object.

Usage

```
## S3 method for class 'textmodel'
summary(object, ...)

## S3 method for class 'textmodelfitted'
summary(object, ...)

## S3 method for class 'textmodelpredicted'
summary(object, ...)
```

Arguments

object	textmodel object to be summarized
...	additional arguments to print

syllableCounts *A named list mapping words to counts of their syllables*

how about just syllables("mytext"), make the data object .syllableCounts.Rdata

Description

A named list mapping words to counts of their syllables, generated from the CMU pronunciation dictionary

References

http://www.speech.cs.cmu.edu/cgi-bin/cmudict

Examples

```
data(syllableCounts)
syllableCounts["sixths"]
syllableCounts["onomatopeia"]
```

`textmodel` *fit a text model*

Description

Fit a text model to a dfm. Creates an S3 object of `textmodelfitted` class, which will also have an S3 class stamp for the specific model fitted. The exact contents of this fitted model will depend on which model was called (see `model` types below).

Usage

```
textmodel(x, y = NULL, data = NULL, model = c("wordscores", "NB",
  "wordfish", "lda", "ca"), ...)

## S4 method for signature 'dfm,ANY,missing,character'
textmodel(x, y = NULL, data = NULL,
  model = c("wordscores", "NB", "wordfish", "lda", "ca"), ...)

## S4 method for signature 'formula,missing,dfm,character'
textmodel(x, y = NULL,
  data = NULL, model = c("wordscores", "NB", "wordfish", "lda", "ca"), ...)
```

Arguments

`x`	a quanteda dfm object containing feature counts by document
`y`	for supervised models, a vector of class labels or values for training the model, with `NA` for documents to be excluded from the training set; for unsupervised models, this will be left NULL
`data`	dfm or data.frame from which to take the formula
`model`	the model type to be fit. Currently implemented methods are:
	`wordscores` Fits the "wordscores" model of Laver, Benoit, and Garry (2003). Options include the original linear scale of LBG or the logit scale proposed by Beauchamps (2001). See textmodel_wordscores.
	`NB` Fits a Naive Bayes model to the dfm, with options for smoothing, setting class priors, and a choice of multinomial or binomial probabilities.
`...`	additional arguments to be passed to specific model types
`formula`	An object of class formula of the form `y ~ x1 + x2 + ...`. (Interactions are not currently allowed for any of the models implemented.) The `x` variable(s) is typically a dfm, and the `y` variable a vector of class labels or training values associated with each document.

Value

a `textmodel` class list, containing the fitted model and additional information specific to the model class. See the methods for specific models, e.g. textmodel_wordscores, textmodel_NB, etc.

Class hierarchy

Here will go the description of the class hierarchy that governs dispatch for the predict, print, summary methods, since this is not terribly obvious. (Blame it on the S3 system.)

See Also

textmodel_wordscores, textmodel_NB, textmodel_wordfish, textmodel_lda, textmodel

Examples

```
data(ie2010Corpus, package="quantedaData")

ieDfm <- dfm(ie2010Corpus, verbose=FALSE)
refscores <- c(rep(NA, 4), -1, 1, rep(NA, 8))
ws <- textmodel(ieDfm, refscores, model="wordscores", smooth=1)
# alternative formula notation - but slower
wsform <- textmodel(refscores ~ . - 1, data=ieDfm, model="wordscores", smooth=1)
identical(ws$pi, wsform$pi)  # compare wordscores from the two models
bs <- textmodel(ieDfm[5:6,], refscores[5:6], model="wordscores", scale="logit", smooth=1)
plot(ws$pi, bs$pi, xlim=c(-1, 1), xlab="Linear word score", ylab="Logit word score")

# prediction method for wordscores
predict(ws, ieDfm, rescaling="mv")

# wordfish
wf <- textmodel(ieDfm, model="wordfish", dir=c(2,1))
```

---

textmodel2 *fit a text model*

Description

Fit a text model to a dfm. Creates an object of virtual class textmodel_fitted-class, whose exact properties (slots and methods) will depend on which model was called (see `model` types below).

Usage

```
textmodel2(x, y = NULL, data = NULL, model = c("wordscores", "NB",
  "wordfish", "lda", "ca"), ...)

## S4 method for signature 'dfm,ANY,missing,character'
textmodel2(x, y = NULL, data = NULL,
  model = c("wordscores", "NB", "wordfish", "lda", "ca"), ...)

## S4 method for signature 'formula,missing,dfm,character'
textmodel2(x, y = NULL,
  data = NULL, model = c("wordscores", "NB", "wordfish", "lda", "ca"), ...)
```

Arguments

`x`	a quanteda dfm object containing feature counts by document
`y`	for supervised models, a vector of class labels or values for training the model, with `NA` for documents to be excluded from the training set; for unsupervised models, this will be left NULL
`data`	dfm or data.frame from which to take the formula
`model`	the model type to be fit. Currently implemented methods are:

`wordscores` Fits the "wordscores" model of Laver, Benoit, and Garry (2003). Options include the original linear scale of LBG or the logit scale proposed by Beauchamps (2001). See textmodel_wordscores.

`NB` Fits a Naive Bayes model to the dfm, with options for smoothing, setting class priors, and a choice of multinomial or binomial probabilities.

`...` additional arguments to be passed to specific model types

`formula` An object of class formula of the form `y ~ x1 + x2 + ...`. (Interactions are not currently allowed for any of the models implemented.) The `x` variable(s) is typically a dfm, and the `y` variable a vector of class labels or training values associated with each document.

Value

a `textmodel` class list, containing the fitted model and additional information specific to the model class. See the methods for specific models, e.g. textmodel_wordscores, textmodel_NB, etc.

Class hierarchy

Here will go the description of the class hierarchy that governs dispatch for the predict, print, summary methods, since this is not terribly obvious. (Blame it on the S3 system.)

See Also

`textmodel_wordscores`, `textmodel_NB`, `textmodel_wordfish`, `textmodel_lda`, `textmodel`

Examples

```
data(ie2010Corpus, package="quantedaData")

ieDfm <- dfm(ie2010Corpus, verbose=FALSE)
refscores <- c(rep(NA, 4), -1, 1, rep(NA, 8))
ws <- textmodel2(ieDfm, refscores, model="wordscores", smooth=1)
# alternative formula notation - but slower
wsform <- textmodel2(refscores ~ . - 1, data=ieDfm, model="wordscores", smooth=1)
identical(ws@Sw, wsform@Sw)  # compare wordscores from the two models
bs <- textmodel2(ieDfm[5:6,], refscores[5:6], model="wordscores", scale="logit", smooth=1)
plot(ws@Sw, bs@Sw, xlim=c(-1, 1), xlab="Linear word score", ylab="Logit word score")
```

`textmodel2_wordscores` *Wordscores text model*

Description

`textmodel_wordscores` implements Laver, Benoit and Garry's (2003) wordscores method for scaling of a single dimension. This can be called directly, but the recommended method is through `textmodel`.

Usage

```
textmodel2_wordscores(data, scores, scale = c("linear", "logit"),
  smooth = 0)

## S3 method for class 'textmodel_wordscores_fitted'
predict(object, newdata = NULL,
  rescaling = "none", level = 0.95, verbose = TRUE, ...)

## S3 method for class 'textmodel_wordscores_fitted'
print(x, n = 30L, ...)

show.textmodel_wordscores_fitted(x, n = 30L, ...)

## S3 method for class 'textmodel_wordscores_predicted'
print(x, ...)
```

Arguments

data	the dfm on which the model will be fit. Does not need to contain only the training documents, since the index of these will be matched automatically.
scores	vector of training scores associated with each document identified in refData
scale	classic LBG linear posterior weighted word class differences, or logit scale of log posterior differences
smooth	a smoothing parameter for word counts; defaults to zero for the to match the LBG (2003) method.
object	the fitted wordscores textmodel on which prediction will be made
newdata	dfm on which prediction should be made
rescaling	none for "raw" scores; lbg for LBG (2003) rescaling; or mv for the rescaling proposed by Martin and Vanberg (2007). (Note to authors: Provide full details here in documentation.)
level	probability level for confidence interval width
verbose	If TRUE, output status messages
...	additional argumennts passed to other functions
x	for print method, the object to be printed
n	max rows of dfm to print

Value

predict.wordscores returns a data.frame whose rows are the documents fitted and whose columns contain the scored textvalues, with the number of columns depending on the options called (for instance, how many rescaled scores, and whether standard errors were requested.) (Note: We may very well change this soon so that it is a list similar to other existing fitted objects.)

Author(s)

Kenneth Benoit

Ken Benoit, borrowed in places from Will Lowe, who probably borrowed from me at some early stage.

References

Laver, Michael, Kenneth R Benoit, and John Garry. 2003. "Extracting Policy Positions From Political Texts Using Words as Data." American Political Science Review 97(02): 311-31; Beauchamp, N. 2012. "Using Text to Scale Legislatures with Uninformative Voting." New York University Mimeo.; Martin, L W, and G Vanberg. 2007. "A Robust Transformation Procedure for Interpreting Political Text." Political Analysis 16(1): 93-100.

LBG (2003); Martin and Vanberg (2007)

Examples

```
data(LBGexample, package="quantedaData")
ws <- textmodel2(LBGexample, c(seq(-1.5, 1.5, .75), NA), model="wordscores")
ws
predict(ws)
# same as:
ws2 <- textmodel2_wordscores(LBGexample, c(seq(-1.5, 1.5, .75), NA))
print(ws2)
wsp2 <- predict(ws2)
print(wsp2)
```

---

textmodel_ca *correspondence analysis of a document-feature matrix*

Description

`textmodel_ca` implements correspondence analysis scaling on a dfm. Currently the method is a wrapper to ca.matrix in the **ca** package.

Usage

```
textmodel_ca(data, smooth = 0, ...)
```

Arguments

`data`	the dfm on which the model will be fit
`smooth`	a smoothing parameter for word counts; defaults to zero.
`...`	additional arguments passed to ca.matrix

Author(s)

Kenneth Benoit

Examples

```
library(quantedaData)
data(ie2010Corpus)
ieDfm <- dfm(ie2010Corpus)
wca <- textmodel_ca(ieDfm)
summary(wca)
```

textmodel_fitted-class *the fitted textmodel classes*

Description

The textmodel virtual class is a parent class for more specific fitted text models, which are the result of a quantitative text analysis applied to a document-feature matrix.

Details

Available types currently include...

Slots

dfm a dfm-class document-feature matrix

textmodel_lda *latent Dirichlet allocation text model*

Description

textmodel_lda estimates the parameters of Blei et. al. (2003) This function is a wrapper to LDA and CTM in **topicmodels**.

Usage

```
textmodel_lda(x, model = c("lda", "ctm", "stm"), k, smooth = 0,
  meta = NULL, ...)
```

Arguments

x	the dfm on which the model will be fit
model	the class of lda-type model to fit. lda for classic latent Dirichlet allocation; ctm for Blei and Rafferty's (2007) correlated topic model.
k	number of topics
smooth	a smoothing parameter for word counts, defaults to 0
meta	metadata passed to stm if fitting a structural topic model
...	additional arguments passed to LDA

Author(s)

Kenneth Benoit

References

Blei, David M., Andrew Y. Ng, and Michael I. Jordan. 2003. "Latent Dirichlet Allocation." *The Journal of Machine Learning Research* 3: 993-1022.

Blei, David M, and John D. Lafferty. 2007. "A Correlated Topic Model of Science." *The Annals of Applied Statistics* 1(1): 17-35.

Roberts, M., Stewart, B., Tingley, D., and Airoldi, E. (2013) "The structural topic model and applied social science." In *Advances in Neural Information Processing Systems Workshop on Topic Models: Computation, Application, and Evaluation*. http://goo.gl/uHkXAQ

Examples

```
## Not run:
data(sotuCorp, package="quantedaData")
SOTUCorpus <- sotuCorp
presDfm <- dfm(subset(SOTUCorpus, year>1960), stopwords=TRUE, stem=TRUE)
presDfm <- trimdfm(presDfm, minCount=5, minDoc=3)
presLDA <- textmodel_lda(presDfm, k=10)
#require(topicmodels)  # need this to access methods below
terms(presLDA, k=10)  # top 10 terms in each topic
topics(presLDA)       # dominant topics for each document
presCTM <- textmodel_lda(presDfm, model="ctm", k=10)
terms(presCTM, k=10)  # top 10 terms in each topic
topics(presCTM)       # dominant topics for each document

# fit a structural topic model
if (require(stm)) {
  gadarianCorpus <- corpus(gadarian$open.ended.response, docvars=gadarian[, 1:3],
                      source="From stm package, from Gadarian and Albertson (forthcoming)")
    gadarianDfm <- dfm(gadarianCorpus, stopwords=TRUE, stem=TRUE)
    gadarianSTM <- textmodel_lda(gadarianDfm, "stm", k=3,
                                 prevalence = ~treatment + s(pid_rep),
                                 data = docvars(gadarianCorpus))
    summary(gadarianSTM)
}

## End(Not run)
```

---

textmodel_NB *Naive Bayes classifier for texts*

Description

Currently working for vectors of texts – not specially defined for a dfm.

Usage

```
textmodel_NB(x, y, smooth = 1, prior = "uniform",
  distribution = "multinomial", ...)
```

Arguments

x	the dfm on which the model will be fit. Does not need to contain only the training documents.
y	vector of training labels associated with each document identified in train. (These will be converted to factors if not already factors.)
smooth	smoothing parameter for feature counts by class
prior	prior distribution on texts, see details
distribution	count model for text features, can be multinomial or Bernoulli
...	more arguments passed through

Details

This naive Bayes model works on word counts, with smoothing.

Value

A list of return values, consisting of:

call	original function call
PwGc	probability of the word given the class (empirical likelihood)
Pc	class prior probability
PcGw	posterior class probability given the word
Pw	baseline probability of the word
data	list consisting of x training class, and y test class
distribution	the distribution argument
prior	argument passed as a prior
smooth	smoothing parameter

Author(s)

Kenneth Benoit

Examples

```
## Example from 13.1 of _An Introduction to Information Retrieval_
trainingset <- matrix(c(1, 2, 0, 0, 0, 0,
                        0, 2, 0, 0, 1, 0,
                        0, 1, 0, 1, 0, 0,
                        0, 1, 1, 0, 0, 1,
                        0, 3, 1, 0, 0, 1),
                      ncol=6, nrow=5, byrow=TRUE,
                      dimnames = list(docs=paste("d", 1:5, sep=""),
                                      features=c("Beijing", "Chinese",  "Japan", "Macao",
                                         "Shanghai", "Tokyo")))
trainingclass <- factor(c("Y", "Y", "Y", "N", NA), ordered=TRUE)
## replicate IIR p261 prediction for test set (document 5)
nb.p261 <- textmodel_NB(as.dfm(trainingset), trainingclass, smooth=1, prior="docfreq")
print(unclass(predict(nb.p261)))
```

`textmodel_wordfish` *Poisson scaling text model*

Description

`textmodel_wordfish` implements Slapin and Proksch's (2008) Poisson scaling model, also known as "wordfish", for a single dimension. This function is a wrapper to wordfish by Will Lowe and to MCMCirtPoisson1d.

Usage

```
textmodel_wordfish(data, method = c("mml", "mcmc"), smooth = 0, ...)
```

Arguments

`data`	the dfm on which the model will be fit. Does not need to contain only the training documents, since the index of these will be matched automatically.
`method`	method for estimating the model. `mml` for Lowe's implementation of marginal maximum likelihood in **austin**; `mcmc` for a Bayesian-MCMC method implemented in JAGS.
`smooth`	a smoothing parameter for word counts; defaults to zero for the to match the LBG (2003) method.
`...`	additional arguments passed to wordfish or to

Author(s)

Kenneth Benoit

References

Benoit, Kenneth and Thomas Daubler. 2014. "Putting Text in Context: How to Estimate Better Left-Right Positions by Scaling Party Manifesto Data." LSE and University of Mannheim manuscript.

Slapin, Jonathan B, and Sven-Oliver Proksch. 2008. "A Scaling Model for Estimating Time-Series Party Positions From Texts." American Journal of Political Science 52(3): 705-22.

Examples

```
library(quantedaData)
data(ie2010Corpus)
ieDfm <- dfm(ie2010Corpus)
wf <- textmodel_wordfish(ieDfm, dir=c(2,1))
summary(wf)
```

`textmodel_wordscores` *Wordscores text model*

Description

`textmodel_wordscores` implements Laver, Benoit and Garry's (2003) wordscores method for scaling of a single dimension. This can be called directly, but the recommended method is through `textmodel`.

Usage

```
textmodel_wordscores(data, scores, scale = c("linear", "logit"), smooth = 0)
```

Arguments

`data`	the dfm on which the model will be fit. Does not need to contain only the training documents, since the index of these will be matched automatically.
`scores`	vector of training scores associated with each document identified in `refData`
`scale`	classic LBG linear posterior weighted word class differences, or logit scale of log posterior differences
`smooth`	a smoothing parameter for word counts; defaults to zero for the to match the LBG (2003) method.

Author(s)

Kenneth Benoit

References

Laver, Michael, Kenneth R Benoit, and John Garry. 2003. "Extracting Policy Positions From Political Texts Using Words as Data." American Political Science Review 97(02): 311-31; Beauchamp, N. 2012. "Using Text to Scale Legislatures with Uninformative Voting." New York University Mimeo.; Martin, L W, and G Vanberg. 2007. "A Robust Transformation Procedure for Interpreting Political Text." Political Analysis 16(1): 93-100.

Examples

```
data(LBGexample, package="quantedaData")
ws <- textmodel(LBGexample, c(seq(-1.5, 1.5, .75), NA), model="wordscores")
ws
# same as:
textmodel_wordscores(LBGexample, c(seq(-1.5, 1.5, .75), NA))
predict(ws)
```

texts *get or set corpus texts*

Description

Get or replace the texts in a quanteda corpus object.

Usage

```
texts(corp)

texts(corp) <- value
```

Need to check and enforce length for assignment, including preventing recycling.

Arguments

corp	A quanteda corpus object
value	character vector of the new texts

Value

For texts, a character vector of the texts in the corpus.

For texts <-, the corpus with the updated texts.

Examples

```
texts(inaugCorpus)[1]
sapply(texts(inaugCorpus), nchar)  # length in characters of the inaugual corpus texts

## this doesn't work yet - need to overload `[` for this replacement function
# texts(inaugTexts)[55] <- "GW Bush's second inaugural address, the condensed version."
```

tokenize *tokenize a set of texts*

Description

Tokenize the texts from a character vector or from a corpus.

Usage

```
tokenize(x, ...)

## S3 method for class 'character'
tokenize(x, simplify = FALSE, sep = " ", ...)

## S3 method for class 'corpus'
tokenize(x, ...)
```

Arguments

x	The text(s) or corpus to be tokenized
...	additional arguments passed to clean
simplify	If TRUE, return a character vector of tokens rather than a list of length ndoc(texts), with each element of the list containing a character vector of the tokens corresponding to that text.
sep	by default, tokenize expects a "white-space" delimiter between tokens. Alternatively, sep can be used to specify another character which delimits fields.

Value

A list of length ndoc(x) of the tokens found in each text.

A list of length ndoc(texts) of the tokens found in each text.

Examples

```
# same for character vectors and for lists
tokensFromChar <- tokenize(inaugTexts)
tokensFromCorp <- tokenize(inaugCorpus)
identical(tokensFromChar, tokensFromCorp)
str(tokensFromChar)
# returned as a list
head(tokenize(inaugTexts[57])[[1]], 10)
# returned as a character vector using simplify=TRUE
head(tokenize(inaugTexts[57], simplify=TRUE), 10)

# demonstrate some options with clean
head(tokenize(inaugTexts[57], simplify=TRUE, lower=FALSE), 30)
```

---

topfeatures *list the most frequent features*

Description

List the most frequently occuring features in a dfm

Usage

```
topfeatures(x, ...)

## S3 method for class 'dfm'
topfeatures(x, n = 10, decreasing = TRUE, ci = 0.95, ...)

## S3 method for class 'dgCMatrix'
topfeatures(x, n = 10, decreasing = TRUE, ...)
```

Arguments

x	the object whose features will be returned
...	additional arguments passed to other methods
n	how many top features should be returned
decreasing	If TRUE, return the n most frequent features, if FALSE, return the n least frequent features
ci	~~confidence interval from 0-1.0 for use if dfm is resampled~~

Value

A named numeric vector of feature counts, where the names are the feature labels.

Examples

```
topfeatures(dfm(inaugCorpus))
topfeatures(dfm(inaugCorpus, stopwords=TRUE))
# least frequent features
topfeatures(dfm(inaugCorpus), decreasing=FALSE)
```

---

trim *Trim a dfm using threshold-based or random feature selection*

---

behaves differently for S3 dense dfm: not sorted.

Description

Returns a document by feature matrix reduced in size based on document and term frequency, and/or subsampling.

Usage

```
trim(x, minCount = 1, minDoc = 1, nsample = NULL, verbose = TRUE)

## S4 method for signature 'dfm'
trim(x, minCount = 1, minDoc = 1, nsample = NULL,
  verbose = TRUE)
```

Arguments

x	document-feature matrix of dfm-class
minCount	minimum feature count
minDoc	minimum number of documents in which a feature appears
nsample	how many features to retain (based on random selection)
verbose	print messages

Value

A dfm-class object reduced in features

Author(s)

Ken Benoit, inspired by code by Will Lowe (see trim)

Examples

```
dtm <- dfm(inaugCorpus)
dim(dtm)
dtmReduced <- trim(dtm, minCount=10, minDoc=2) # only words occuring >=5 times and in >=2 docs
dim(dtmReduced)
topfeatures(dtmReduced, decreasing=FALSE)
dtmSampled <- trim(dtm, minCount=20, nsample=50)  # sample 50 words over 20 count
dtmSampled # 57 x 50 words
topfeatures(dtmSampled)
```

---

trimdfm *Trim a dfm based on a subset of features and words*

Description

Returns a document by feature matrix reduced in size based on document and term frequency, and/or subsampling.

Usage

```
trimdfm(x, minCount = 1, minDoc = 1, minTotal = 0, sample = NULL,
  keep = NULL, verbose = TRUE)
```

Arguments

x	document-feature matrix created by dfm
minCount	minimum feature count
minDoc	minimum number of documents in which a feature appears
minTotal	minimum total feature threshold to retain a document
sample	how many features to retain (based on random selection)
keep	regular expression specifying which features to keep
verbose	print messages

Value

A dfm object reduced in size.

Author(s)

Ken Benoit adapted from code originally by Will Lowe (see trim)

Examples

```
dtm <- dfm(inaugCorpus)
dim(dtm)
dtmReduced <- trimdfm(dtm, minCount=10, minDoc=2) # only words occuring >=5 times and in >=2 docs
dim(dtmReduced)
dtmReduced <- trimdfm(dtm, keep="^nation|^citizen|^union$")
topfeatures(dtmReduced, NULL)
dtmSampled <- trimdfm(dtm, sample=200)  # top 200 words
dim(dtmSampled)  # 196 x 200 words
```

uk2010immig *Immigration-related sections of 2010 UK party manifestos*

immigrationTexts

Description

Extracts from the election manifestos of 9 UK political parties from 2010, related to immigration or asylum-seekers.

Format

A named character vector of plain ASCII texts

Examples

```
data(uk2010immig)
uk2010immigCorpus <- corpus(uk2010immig, docvars=list(party=names(uk2010immig)))
language(uk2010immigCorpus) <- "english"
encoding(uk2010immigCorpus) <- "UTF-8"
summary(uk2010immigCorpus)
```

weight *Weight the feature frequencies in a dfm by various methods*

Description

Returns a document by feature matrix with the feature frequencies weighted according to one of several common methods.

Usage

```
weight(x, ...)

## S4 method for signature 'dfm'
weight(x, type = c("frequency", "relFreq", "relMaxFreq",
  "logFreq", "tfidf"), smooth = 0, normalize = TRUE, verbose = TRUE, ...)

tf(x)

tfidf(x)
```

too many functions here??

```
smoother(x, smooth)

weighting(object)

## S4 method for signature 'dfm'
weighting(object)
```

Arguments

x	document-feature matrix created by dfm
...	not currently used
type	The weighting function to aapply to the dfm. One of: • normTf - Length normalization: dividing the frequency of the feature by the length of the document) • logTf - The natural log of the term frequency • tf-idf - Term-frequency * inverse document frequency. For a full explanation, see, for example, (http://nlp.stanford.edu/IR-book/html/htmledition/term-frequency-and-weighting-1.html). This implementation will not return negative values. • maxTf - The term frequency divided by the frequency of the most frequent term in the document • ppmi - Positive Pointwise Mutual Information
smooth	amount to apply as additive smoothing to the document-feature matrix prior to weighting, default is 0.5, set to smooth=0 for no smoothing.
normalize	if TRUE (default) then normalize the dfm by relative term frequency prior to computing tfidf
verbose	if TRUE output status messages
object	the dfm object for accessing the weighting setting

Details

tf is a shortcut for weight(x, "relFreq")

tfidf is a shortcut for weight(x, "tfidf")

smoother is a shortcut for weight(x, "frequency", smooth)

weighting queries (but cannot set) the weighting applied to the dfm.

Value

The dfm with weighted values

weighting returns a character object describing the type of weighting applied to the dfm.

Author(s)

Paul Nulty and Kenneth Benoit

References

Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schutze. Introduction to information retrieval. Vol. 1. Cambridge: Cambridge university press, 2008.

Examples

```
dtm <- dfm(inaugCorpus)
x <- apply(dtm, 1, function(tf) tf/max(tf))
topfeatures(dtm)
normDtm <- weight(dtm)
topfeatures(normDtm)
```

```
maxTfDtm <- weight(dtm, type="relMaxFreq")
topfeatures(maxTfDtm)
logTfDtm <- weight(dtm, type="logFreq")
topfeatures(logTfDtm)
tfidfDtm <- weight(dtm, type="tfidf")
topfeatures(tfidfDtm)

# combine these methods for more complex weightings, e.g. as in Section 6.4 of
# Introduction to Information Retrieval
logTfDtm <- weight(dtm, type="logFreq")
wfidfDtm <- weight(logTfDtm, type="tfidf", normalize=FALSE)
```

~~wordfishMCMC~~ ~~*Bayesian MCMC version of the "wordfish" Poisson scaling model*~~

Description

wordfishMCMC implements a flexible, Bayesian model estimated in JAGS using MCMC. It is based on the implementation of wordfish from the austin package. Options include specifying a model for alpha using document-level covariates, and partitioning the word parameters into different subsets, for instance, countries.

Usage

```
wordfishMCMC(dtm, dir = c(1, 2), control = list(sigma = 3, startparams =
  NULL), alphaModel = c("free", "logdoclength", "modelled"),
  alphaFormula = NULL, alphaData = NULL, wordPartition = NULL,
  betaPartition = FALSE, wordConstraints = NULL, verbose = TRUE,
  PoissonGLM = FALSE, nChains = 1, nAdapt = 100, nUpdate = 300,
  nSamples = 100, nThin = 1, ...)
```

Arguments

dtm	The document-term matrix. Ideally, documents form the rows of this matrix and words the columns, although it should be correctly coerced into the correct shape.
dir	A two-element vector, enforcing direction constraints on theta and beta, which ensure that theta[dir[1]] < theta[dir[2]]. The elements of dir will index documents.
control	list specifies options for the estimation process. These are: tol, the proportional change in log likelihood sufficient to halt estimatioe, sigma the standard deviation for the beta prior in poisson form, and startparams a previously fitted wordfish model. verbose generates a running commentary during estimation. See wordfish.
alphaModel	free means the α_i is entirely estimated; logdoclength means the alpha is predicted with an expected value equal to the log of the document length in words, similar to an offset in a Poisson model with variable exposure; modelled allows you to specify a formula and covariates for α_i using alphaFormula and alphaData.
alphaFormula	Model formula for hierarchical model predicting α_i.
alphaData	Data to form the model matrix for the hierarchical model predicting α_i.

wordPartition	A vector equal in length to the documents that specifies a unique value partitioning the word parameters. For example, alpha could be a Boolean variable for EU to indicate that a document came from a country outside the EU or inside the EU. Or, it could be a factor variable indicating the name of the country (as long as there are multiple documents per country). Internally, wordPartition is coerced to a factor. NULL indicates that no paritioning of the word-level parameters will take place (default).
betaPartition	Boolean indicating that the β parameter should also be partitioned according to wordPartition.
wordConstraints	An index with a minimim length of 1, indicating which words will be set equal across the wordPartition factors. NULL if is.null(wordPartition) (default).
verbose	Turn this on for messages. Default is TRUE.
PoissonGLM	Boolean denoting that the basic model should be estimated where log(alpha) is ~ dflat() as per The BUGS Book pp131-132
nChains	Number of chains to run in JAGS.
nAdapt	Adaptation iterations in JAGS.
nUpdate	Update iterations in JAGS.
nSamples	Number of posterior samples to draw in JAGS.
nThin	Thinning parameter for drawing posterior samples in JAGS.
...	Additional arguments passed through.

Value

An augmented wordfish class object with additional stuff packed in. To be documented.

Author(s)

Kenneth Benoit

Examples

```
## Not run:
data(iebudgets)
# extract just the 2010 debates
iebudgets2010 <- corpus.subset(iebudgets, year==2010)

# create a document-term matrix and set the word margin to the columns
dtm <- create.fvm.corpus(iebudgets2010)
dtm <- wfm(t(dtm), word.margin=2)

# estimate the maximium likelihood wordfish model from austin
iebudgets2010_wordfish <- austin::wordfish(dtm, dir=c(2,1))

# estimate the MCMC model, default values
iebudgets2010_wordfishMCMC <- wordfishMCMC(dtm, dir=c(2,1))

# compare the estimates of \eqn{\theta_i}
plot(iebudgets2010_wordfish$theta, iebudgets2010_wordfishMCMC$theta)

# MCMC with a partition of the word parameters according to govt and opposition
```

```
# (FF and Greens were in government in during the debate over the 2010 budget)
# set the constraint on word partitioned parameters to be the same for "the" and "and"
iebudgets2010_wordfishMCMC_govtopp <-
    wordfishMCMC(dtm, dir=c(2,1),
   wordPartition=(iebudgets2010$attribs$party=="FF" | iebudgets2010$attribs$party=="Green"),
    betaPartition=TRUE, wordConstraints=which(words(dtm)=="the"))

## End(Not run)
```

`wordstem` *stem words*

Description

Apply a stemmer to words. This is a wrapper to wordStem designed to allow this function to be called without loading the entire **SnowballC** package. wordStem uses Dr. Martin Porter's stemming algorithm and the C libstemmer library generated by Snowball.

Usage

```
wordstem(words, language = "porter")
```

Arguments

`words`	a character vector of words whose stems are to be extracted.
`language`	the name of a recognized language, as returned by getStemLanguages, or a two- or three-letter ISO-639 code corresponding to one of these languages (see references for the list of codes)

Value

A character vector with as many elements as there are in the input vector with the corresponding elements being the stem of the word. Elements of the vector are converted to UTF-8 encoding before the stemming is performed, and the returned elements are marked as such when they contain non-ASCII characters.

References

http://snowball.tartarus.org/

http://www.loc.gov/standards/iso639-2/php/code_list.php for a list of ISO-639 language codes

See Also

wordStem

Examples

```
# Simple example
wordstem(c("win", "winning", "winner"))
```

`zipfiles` *unzip a zipped collection of text files and return the directory*

Description

`zipfiles` extracts a set of text files in a zip archives, and returns the name of the temporary directory where they are stored. It can be passed to corpus.directory for import.

Usage

```
zipfiles(zfile = NULL, ...)
```

Arguments

`zfile`	a character string specifying the name (including path) of the zipped file, or a URL naming the file (see example); or NULL to use a GUI to choose a file from disk
`...`	additional arguments passed to unzip

Value

a directory class object containing the unzipped files

Examples

```
## Not run:
# from a zip file on the web
myzipcorp <- corpus(zipfiles("http://kenbenoit.net/files/EUcoalsubsidies.zip"),
                    notes="From some EP debate about coal mine subsidies")
docvars(myzipcorp, "speakername") <- docnames(myzipcorp)
summary(myzipcorp)

# call up interactive user input
myzipcorp <- corpus(zipfiles())

## End(Not run)
```

Index

+,dfmDense,numeric-method *(dfm-class)*, 17
+,dfmSparse,numeric-method *(dfm-class)*, 17
+,numeric,dfmDense-method *(dfm-class)*, 17
+,numeric,dfmSparse-method *(dfm-class)*, 17
+.corpus *(corpus)*, 10
[,dfmDense,index,index,logical-method *(dfm-class)*, 17
[,dfmDense,index,index,missing-method *(dfm-class)*, 17
[,dfmDense,index,missing,logical-method *(dfm-class)*, 17
[,dfmDense,index,missing,missing-method *(dfm-class)*, 17
[,dfmDense,missing,index,logical-method *(dfm-class)*, 17
[,dfmDense,missing,index,missing-method *(dfm-class)*, 17
[,dfmDense,missing,missing,logical-method *(dfm-class)*, 17
[,dfmDense,missing,missing,missing-method *(dfm-class)*, 17
[,dfmSparse,index,index,logical-method *(dfm-class)*, 17
[,dfmSparse,index,index,missing-method *(dfm-class)*, 17
[,dfmSparse,index,missing,logical-method *(dfm-class)*, 17
[,dfmSparse,index,missing,missing-method *(dfm-class)*, 17
[,dfmSparse,missing,index,logical-method *(dfm-class)*, 17
[,dfmSparse,missing,index,missing-method *(dfm-class)*, 17
[,dfmSparse,missing,missing,logical-method *(dfm-class)*, 17
[,dfmSparse,missing,missing,missing-method *(dfm-class)*, 17
[.dfm *(dfm-class)*, 17

as.data.frame,dfm-method *(dfm)*, 14
as.dfm *(dfm)*, 14
as.DocumentTermMatrix, *22*
as.matrix,dfm-method *(dfm)*, 14
austin, *35*

bigrams, 4, *8*

ca.matrix, *63*
changeunits, 5, *48*
clean, 6, *7*, *15*, *47*, *70*
collocations, 7
colSums,dfmSparse-method, 8
compoundWords, 9
corpus, *10*, 10
corpus.directory, *78*
countSyllables, 13
CTM, *64*

data.frame, *11*
describeTexts, 14
dfm, *12*, 14, *15*, *16*, *20–23*, *26*, *41*, *49–52*, *55*, *59–61*, *63*, *70*, *72*, *74*
dfm-class, *14*, *16*, 17, *23*, *64*, *71*
dfm2ldaformat, 20
dfm2stmformat, 21
dfm2tmformat, 21
dfmDense-class *(dfm-class)*, 17
dfmSparse-class *(dfm-class)*, 17
dgCMatrix-class, *19*
dgeMatrix-class, *19*
directory, *10*, *11*, 22, *78*
docfreq, 23
docfreq,dfm,missing-method *(docfreq)*, 23
docfreq,dfm,numeric-method *(docfreq)*, 23
docfreq,dfmDense,missing-method *(docfreq)*, 23
docfreq,dfmDense,numeric-method *(docfreq)*, 23
docfreq,dfmSparse,missing-method *(docfreq)*, 23
docfreq,dfmSparse,numeric-method *(docfreq)*, 23
docnames, 23
docnames<- *(docnames)*, 23

document-feature matrix object, *52*
DocumentTermMatrix, *21*
docvars, *11, 12*, 24
docvars<- *(*docvars*)*, 24

Encoding, *25*, *29*, *31*
encoding, *12*, 25
encoding<- *(*encoding*)*, 25
excel, 26

features, 26
file, *22*
flatten.dictionary, 27
formula, *59*, *61*

getFBpage, 28
getRootFileNames, 29
getTextDir, 29
getTextDirGui, 30
getTextFiles, 30
getTimeline, 31
getTweets, 32

iconv, *11*
iconvlist, *11*, *25*, *44, 45*
inaugCorpus, 33
inaugTexts *(*inaugCorpus*)*, 33
is.corpus *(*corpus*)*, 10
is.dfm *(*dfm*)*, 14
is.resampled *(*resample*)*, 46

json, 33

kwic, 34

language, *12*, 35
language<- *(*language*)*, 35
LDA, *64*
lda.collapsed.gibbs.sampler, *20*

Matrix, *8*, *15*
Matrix-class, *17*, *20*
MCMCirtPoisson1d, 35, *36*, *67*
metacorpus, *12*, 37
metacorpus<- *(*metacorpus*)*, 37
metadoc, *11, 12*, *25*, *35*, 38
metadoc<- *(*metadoc*)*, 38

ndoc, 39, *70*
nfeature *(*ndoc*)*, 39
ngrams, *8*, 39
nresample, 40

parseTweets, *33*
plot.dfm, 41
pr_DB, *50*
predict.naivebayes *(*predict.textmodel*)*, 42
predict.textmodel, 42
predict.textmodel_wordscores_fitted *(*textmodel2_wordscores*)*, 61
predict.textmodelfitted *(*predict.textmodel*)*, 42
predict.wordscores *(*predict.textmodel*)*, 42
print, *58*
print,dfmDense-method *(*print,dfmSparse-method*)*, 43
print,dfmSparse-method, 43
print.dfm *(*print,dfmSparse-method*)*, 43
print.settings *(*settings*)*, 49
print.textmodel, 44
print.textmodel_wordscores_fitted *(*textmodel2_wordscores*)*, 61
print.textmodel_wordscores_predicted *(*textmodel2_wordscores*)*, 61
print.textmodelfitted *(*print.textmodel*)*, 44
print.textmodelpredicted *(*print.textmodel*)*, 44

quanteda-package, 4, *33*

readLIWCdict, 44
readWStatDict, 45
regex, *47, 48*
regular expression, *6*, *11*, *29*
resample, 46

segment, *5*, *46*
segment *(*segmentSentence*)*, 47
segmentParagraph *(*segmentSentence*)*, 47
segmentSentence, 47
settings, *12*, *20*, *43*, 49
settings<- *(*settings*)*, 49
settingsInitialize, 50
show.textmodel_wordscores_fitted *(*textmodel2_wordscores*)*, 61
simil, *50*
similarity, 50
similarity,dfm,index-method *(*similarity*)*, 50
simple triplet matrix, *21*
smooth, *20*
smoother *(*weight*)*, 73
sort.dfm, 51
statLexdiv, 52

stm, *21*, *64*
stopwords, 54, *55*
stopwordsGet, *15*, *55*, 55, *56*
stopwordsRemove, 55
strheight, *41*
strwidth, *41*
subset.corpus, 56
summary.corpus, 57
summary.textmodel, 58
summary.textmodelfitted
(summary.textmodel), 58
summary.textmodelpredicted
(summary.textmodel), 58
syllableCounts, 58

t,dfm-method (dfm-class), 17
t,dfmDense-method (dfm-class), 17
t,dfmSparse-method (dfm-class), 17
text, *41*
textmodel, *42*, 59, *60, 61*, *68*
textmodel,dfm,ANY,missing,character-method
(textmodel), 59
textmodel,formula,missing,dfm,character-method
(textmodel), 59
textmodel2, 60
textmodel2,dfm,ANY,missing,character-method
(textmodel2), 60
textmodel2,formula,missing,dfm,character-method
(textmodel2), 60
textmodel2_wordscores, 61
textmodel_ca, 63
textmodel_fitted-class, *60*, 64
textmodel_lda, *60, 61*, 64
textmodel_NB, *59–61*, 65
textmodel_wordfish, *60, 61*, 67
textmodel_wordscores, *59–61*, 68
textmodel_wordscores_fitted-class
(textmodel_fitted-class), 64
textmodel_wordscores_predicted-class
(textmodel_fitted-class), 64
texts, *12*, 69
texts<- (texts), 69
tf (weight), 73
tfidf (weight), 73
tokenise (tokenize), 69
tokenize, *4*, *40*, 69
topFeatures (topfeatures), 70
topfeatures, 70
trim, *71*, 71, *72*
trim,dfm-method (trim), 71
trimdfm, 72

uk2010immig, 73
unlist, *37*
unzip, *78*

VCorpus, *10*, *12*

weight, *20*, 73
weight,dfm-method (weight), 73
weighting (weight), 73
weighting,dfm-method (weight), 73
wordcloud, *41*
wordfish, *35, 36*, *67*, *75*
wordfishMCMC, 75
wordStem, *77*
wordstem, 77

zipfiles, *10*, 78